Red Hat Enterprise Linux 7

仮想化のチューニングと最適化ガイド

RHEL のホストシステムおよび仮想化ゲストに KVM パフォーマンス機能を使用

Last Updated: 2021-01-06

Red Hat Enterprise Linux 7 仮想化のチューニングと最適化ガイド

RHEL のホストシステムおよび仮想化ゲストに KVM パフォーマンス機能を使用

Jiri Herrmann
Red Hat Customer Content Services
jherrman@redhat.com

Yehuda Zimmerman
Red Hat Customer Content Services
yzimmerm@redhat.com

Dayle Parker
Red Hat Customer Content Services

Scott Radvan
Red Hat Customer Content Services

Red Hat Subject Matter Experts

法律上の通知

概要

Red Hat Enterprise Linux 仮想化のチューニングと最適化ガイドでは、KVM および仮想化のパフォーマンスを説明します。KVM のパフォーマンス関連の機能やオプションをホストのシステムや仮想化ゲストに活かすためのヒントや推奨事項なども記載しています。

目次

第1章 はじめに 3
1.1. 仮想システムでパフォーマンスの最適化が必要な理由 3
1.2. KVM のパフォーマンスについて 3
1.3. 仮想化のパフォーマンス関連の特長および改善点 3

第2章 パフォーマンス監視ツール 6
2.1. PERF KVM 6
2.2. 仮想パフォーマンスの監視ユニット (VPMU: VIRTUAL PERFORMANCE MONITORING UNIT) 7
2.3. 仮想マシンマネージャーでのパフォーマンス監視 8

第3章 VIRT-MANAGER を使用した仮想化パフォーマンスの最適化 20
3.1. オペレーティングシステムの詳細とデバイス 20
3.2. CPU パフォーマンスのオプション 21
3.3. 仮想ディスクパフォーマンスのオプション 24

第4章 TUNED と TUNED-ADM 25

第5章 ネットワーク 27
5.1. ネットワークチューニングのヒント 27
5.2. VIRTIO と VHOST_NET 27
5.3. デバイスの割り当てと SR-IOV 27
5.4. ネットワークチューニングのヒント 28
5.5. ネットワークパケットのバッチ処理 30

第6章 I/O スケジューリング 31
6.1. RED HAT ENTERPRISE LINUX を仮想化ホストとして使用する場合の I/O スケジューリング 31
6.2. RED HAT ENTERPRISE LINUX を仮想化ゲストとして使用する場合の I/O スケジューリング 31

第7章 ブロック I/O 33
7.1. ブロック I/O チューニング 33
7.2. キャッシュ 34
7.3. I/O モード 35
7.4. ブロック I/O のチューニング方法 35

第8章 メモリー 38
8.1. メモリーチューニングのヒント 38
8.2. 仮想マシン上でのメモリーチューニング 38
8.3. KSM (KERNEL SAME-PAGE MERGING) 43

第9章 NUMA 49
9.1. NUMA メモリー割り当てのポリシー 49
9.2. NUMA の自動負荷分散 49
9.3. LIBVIRT の NUMA チューニング 50
9.4. NUMA 対応 KSM (KERNEL SAMEPAGE MERGING) 59

付録A 改訂履歴 61

第1章 はじめに

1.1. 仮想システムでパフォーマンスの最適化が必要な理由

KVM 仮想化では、ゲストはホストマシンではプロセスにより表現されています。これは、処理能力、メモリー、およびホストの他のリソースが、ゲストの仮想ハードウェアの機能や能力を模倣するために使用されていることを意味します。

ただし、ゲストのハードウェアでは、リソースを使用する有効性がホストよりも減る可能性があります。したがって、期待される速度でゲストがそのタスクを実行するためには、ホストリソースに割り当てている量を調整することが必要になる場合があります。さらに、さまざまな種類の仮想ハードウェアでは、オーバーヘッドのレベルが異なる場合があるため、適切な仮想ハードウエアの構成により、ゲストのパフォーマンスでの影響が大きくなる可能性があります。最後に、状況に応じて、具体的な構成により、より効率的にホストのリソースを使用することができるようになります。

1.2. KVM のパフォーマンスについて

KVM は、システムパフォーマンスだけでなｋ、プロセスやスレッドの管理に関連するため、以下に KVM の概要を示します。

- KVM を使用すると、ゲストはホストの Linux プロセスとして実行します。
- 仮想 CPU (vCPU) は通常のスレッドとして実装され、Linux スケジューラーで処理されます。
- ゲストは NUMA や Huge Page などの機能を、カーネルから自動的に継承しません。
- ホスト内でのディスクおよびネットワーク I/O の設定はパフォーマンスに多大な影響を与えます。
- ネットワークトラフィックは一般的にはソフトウェアベースのブリッジを通過します。
- デバイスとそのモデルによっては、その特定のハードウェアのエミュレーションに大きなオーバーヘッドがある可能性があります。

1.3. 仮想化のパフォーマンス関連の特長および改善点

Red Hat Enterprise Linux 7 における仮想化パフォーマンスの改善点

Red Hat Enterprise Linux 7 では、以下の機能により仮想化のパフォーマンスが強化されています。

NUMA の自動負荷分散

NUMA の自動負荷分散により、Red Hat Enterprise Linux 7 ゲストで必要な手動のチューニングなしに、NUMA ハードウェアシステムで実行されるアプリケーションのパフォーマンスを強化できます。NUMA の自動負荷分散は、スレッドやプロセスなどのタスクを、それらがアクセスするメモリーに近い場所に移動します。これにより、ゼロ構成で良好なパフォーマンスが可能になりますが、一部の状況では、より正確なゲスト構成を行うか、CPU およびメモリーにホストのアフィニティーを設定すると、より良い結果が提供されます。

NUMA の自動負荷分散に関する詳細は、「NUMA の自動負荷分散」を参照してください。

VirtIO モデル

virtio モデルがあるすべての仮想ハードウェアには、そのすべての特殊性を持つハードウェアをエミュレートするオーバーヘッドがありません。VirtIO デバイスには、それらが仮想化環境での使用

のために特別に設計されているため、オーバーヘッドが少なくなりますが、すべてのゲストオペレーティングシステムでこのモデルに対応しているわけではありません。

マルチキュー virtio-net

パケットの送信/受信処理の規模をゲストの利用可能な vCPU の数に合わせて調整できるようにするネットワーク手法です。

マルチキュー virtio-net についての詳細は、「マルチキュー virtio-net」を参照してください。

ブリッジのゼロコピー送信

ゼロコピー送信モードは、ゲストネットワークと外部ネットワーク間での大規模なパケットを送信する際のホスト CPU のオーバーヘッドを、スループットに影響を与えることなく最高 15% 削減します。ブリッジのゼロコピー送信は Red Hat Enterprise Linux 7 仮想マシンで完全にサポートされていますが、デフォルトでは無効にされています。

ゼロコピー送信についての詳細は、「ブリッジのゼロコピー送信」を参照してください。

APIC Virtualization (APICv)

新たな Intel プロセッサーは Advanced Programmable Interrupt Controller (APICv) のハードウェア仮想化機能を提供します。APICv により、ゲストの直接的な APIC アクセスが許可され、APIC による割り込みの待機時間や仮想マシンの終了回数が大幅に削減されるため、仮想化された AMD64 および Intel 64 ゲストのパフォーマンスが強化されます。この機能は新規の Intel プロセッサーでデフォルトで使用され、I/O パフォーマンスを向上させます。

EOI の加速化

仮想 APIC 機能なしに、旧式チップセットの高帯域幅 I/O の割り込み終了 (EOI: End-of-interrupt) 処理を加速化します。

マルチキュー virtio-scsi

virtio-scsi ドライバーのマルチキュー対応により、ストレージのパフォーマンスとスケーラビリティーが強化されます。これにより、他の vCPU に影響を与えることなく、それぞれの仮想 CPU で別個のキューや割り込みを使用できます。

マルチキュー virtio-scsi についての詳細は、「マルチキュー virtio-scsi」を参照してください。

準仮想化された Ticketlock

準仮想化された ticketlock (pvticketlock) は、オーバーサブスクライブされた CPU を持つ Red Hat Enterprise Linux 7 ホスト上で実行される Red Hat Enterprise Linux 7 ゲスト仮想マシンのパフォーマンスを強化します。

準仮想化されたページフォールト

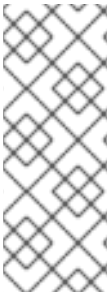
準仮想化されたページフォールトは、ゲストがホストによってスワップアウトされるページへのアクセスを試行する際にゲストに挿入されます。これにより、ホストのメモリーがオーバーコミットされ、ゲストのメモリーがスワップアウトされる場合に KVM ゲストのパフォーマンスが向上します。

準仮想化された時刻関連の vsyscall の最適化

gettimeofday および **clock_gettime** システム呼び出しを **vsyscall** メカニズムによりユーザー領域で実行できます。以前は、これらのシステム呼び出しを実行すると、システムをカーネルモードに切り替えてからユーザー領域に戻す必要がありました。これにより、一部のアプリケーションのパフォーマンスが大幅に向上します。

Red Hat Enterprise Linux における仮想化パフォーマンスの機能

- CPU/カーネル
  - NUMA: Non-Uniform Memory Access (非均一なメモリーアクセス)。詳細は「9章*NUMA*」を参照してください。
  - CFS: Completely Fair Scheduler (完全に公平なスケジューラー)。クラスに焦点を置いた新しいスケジューラーです。
  - RCU: Read Copy Update (リードコピーアップデート)。共有スレッドデータの処理が向上しました。
  - 仮想 CPU (vCPU) の最大数は 160 になります。
- メモリー
  - Huge Page およびメモリー集約型環境での各種の最適化。詳細は「8章*メモリー*」を参照してください。
- ネットワーク
  - vhost-net: カーネルベースの高速 virtIO ソリューション
  - SR-IOV: ネイティブに近いネットワークパフォーマンスレベルを実現するために導入
- ブロック I/O
  - AIO: 他の I/O 動作にオーバーラップするスレッドのサポート
  - MSI: PCI バスデバイスの割り込み生成
  - ディスク I/O スロットリング: ゲストのディスク I/O 要求を制御し、ホストリソースの過剰使用を防ぎます。詳細は、「ディスク I/O スロットリング」を参照してください。



注記

仮想化に関するサポート、制約、および特長などの詳細は、『Red Hat Enterprise Linux 7 仮想化スタートガイド』および以下の URL を参照してください。

https://access.redhat.com/certified-hypervisors

https://access.redhat.com/ja/articles/1520293

第2章 パフォーマンス監視ツール

本章では、ゲストの仮想マシン環境の監視に使用するツールについて説明します。

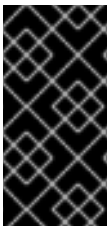
2.1. PERF KVM

ホストからゲストのオペレーティングシステムの統計値を収集し、分析するには、**perf** コマンドに **kvm** オプションを付けて使用します。perf パッケージは **perf** コマンドを提供します。これは、以下のコマンドを実行してインストールできます。

```
# yum install perf
```

ホストで **perf kvm** を使用するには、ゲストの **/proc/modules** ファイルと **/proc/kallsyms** ファイルにアクセスできなければなりません。ホストにファイルを転送してから、そのファイルでレポートを実行するには、手順2.1「ゲストの /proc ファイルをホストにコピーする」を参照してください。

手順2.1 ゲストの /proc ファイルをホストにコピーする



重要

必要なファイルを /proc ディレクトリーから直接コピーしても (**scp** などを使用)、コピーしたファイルは空になります。本セクションでは、まず、ゲストのファイルを一時的な場所に保存し (**cat** コマンドを使用)、その保存先からファイルをホストにコピーして、**perf kvm** で使用する手順を説明しています。

1. **ゲストにログインしてファイルを保存する**
   ゲストにログインして、**/proc/modules** と **/proc/kallsyms** を一時的な場所の **/tmp** に保存します。

   ```
   # cat /proc/modules > /tmp/modules
   # cat /proc/kallsyms > /tmp/kallsyms
   ```

2. **一時ファイルをホストにコピーする**
   ゲストからログオフしたら、今度は次の例に示す **scp** コマンドを実行して一時的な場所に保存したファイルをホストにコピーします。必要に応じてホスト名と TCP ポートをご使用の値に置き換えてください。

   ```
   # scp root@GuestMachine:/tmp/kallsyms guest-kallsyms
   # scp root@GuestMachine:/tmp/modules guest-modules
   ```

   これでゲストからの 2 つのファイル (**guest-kallsyms** と **guest-modules**) がホスト上にコピーされ、**perf kvm** で使用する準備が整いました。

3. **perf kvm でイベントの記録とレポートを実行する**
   前述の手順で入手したファイルを使って、ゲスト内のイベント、ホスト内のイベントのいずれかまたは両方を記録し、レポートできるようになりました。

   次のコマンドの例を実行します。

   ```
   # perf kvm --host --guest --guestkallsyms=guest-kallsyms \
   --guestmodules=guest-modules record -a -o perf.data
   ```

注記

--host と **--guest** の両方をコマンドに使用すると、出力は **perf.data.kvm** というファイル名で保存されます。**--host** だけを使用すると、ファイル名は **perf.data.host** になります。同じように、**--guest** のみを使用すると、そのファイル名は **perf.data.guest** になります。

記録の動作を停止させる場合は Ctrl-C を押します。

4. **イベントをレポートする**
記録のプロセスで取得したファイルを使って出力を新しいファイル「**analyze**」にリダイレクトする例を示します。

```
perf kvm --host --guest --guestmodules=guest-modules report -i perf.data.kvm \
--force > analyze
```

記録したイベントを調べるため、**analyze** ファイルの内容を表示します。

```
# cat analyze


# Events: 7K cycles
#
# Overhead       Command  Shared Object      Symbol
# ........  ............  .................  ........................
#
    95.06%            vi  vi                 [.] 0x48287
     0.61%          init  [kernel.kallsyms]  [k] intel_idle
     0.36%            vi  libc-2.12.so       [.] _wordcopy_fwd_aligned
     0.32%            vi  libc-2.12.so       [.] __strlen_sse42
     0.14%       swapper  [kernel.kallsyms]  [k] intel_idle
     0.13%          init  [kernel.kallsyms]  [k] uhci_irq
     0.11%          perf  [kernel.kallsyms]  [k] generic_exec_single
     0.11%          init  [kernel.kallsyms]  [k] tg_shares_up
     0.10%      qemu-kvm  [kernel.kallsyms]  [k] tg_shares_up

[output truncated...]
```

2.2. 仮想パフォーマンスの監視ユニット (VPMU: VIRTUAL PERFORMANCE MONITORING UNIT)

仮想パフォーマンスの監視ユニット (vPMU) は、ゲスト仮想マシンがどのように機能しているかを示す統計を表示します。

仮想パフォーマンス監視ユニットを使用すると、ユーザーはゲスト仮想マシン内の潜在的なパフォーマンス問題の出所を特定することができます。vPMU は Intel の PMU (Performance Monitoring Unit) をベースとしており、Intel マシンでのみ使用できます。

この機能は、Red Hat Enterprise Linux 6 または Red Hat Enterprise Linux 7 を実行するゲスト仮想マシンでのみサポートされ、デフォルトでは無効にされています。

vPMU がご使用のシステムでサポートされているかどうかを確認するには、以下を実行してホスト CPU 上で **arch_perfmon** フラグをチェックします。

```
# cat /proc/cpuinfo|grep arch_perfmon
```

vPMU を有効にするには、ゲスト XML 内で ***cpu mode*** を **host-passthrough** として指定します。

```
# virsh dumpxml guest_name |grep "cpu mode"
<cpu mode='host-passthrough'>
```

vPMU が有効にされた後に、ゲスト仮想マシンから **perf** コマンドを実行して仮想マシンのパフォーマンス統計を表示します。

2.3. 仮想マシンマネージャーでのパフォーマンス監視

仮想マシンモニターを使用すると、システム上の仮想マシンのパフォーマンス情報を表示できます。仮想マシンマネージャーで表示されるパフォーマンス情報を設定することもできます。

2.3.1. 仮想マシンマネージャーでのパフォーマンス概要の表示

仮想マシンマネージャーを使用して仮想マシンのパフォーマンス概要を表示するには、以下の手順を実施します。

1. 仮想マシンマネージャーのメインウィンドウで表示する仮想マシンを強調表示します。

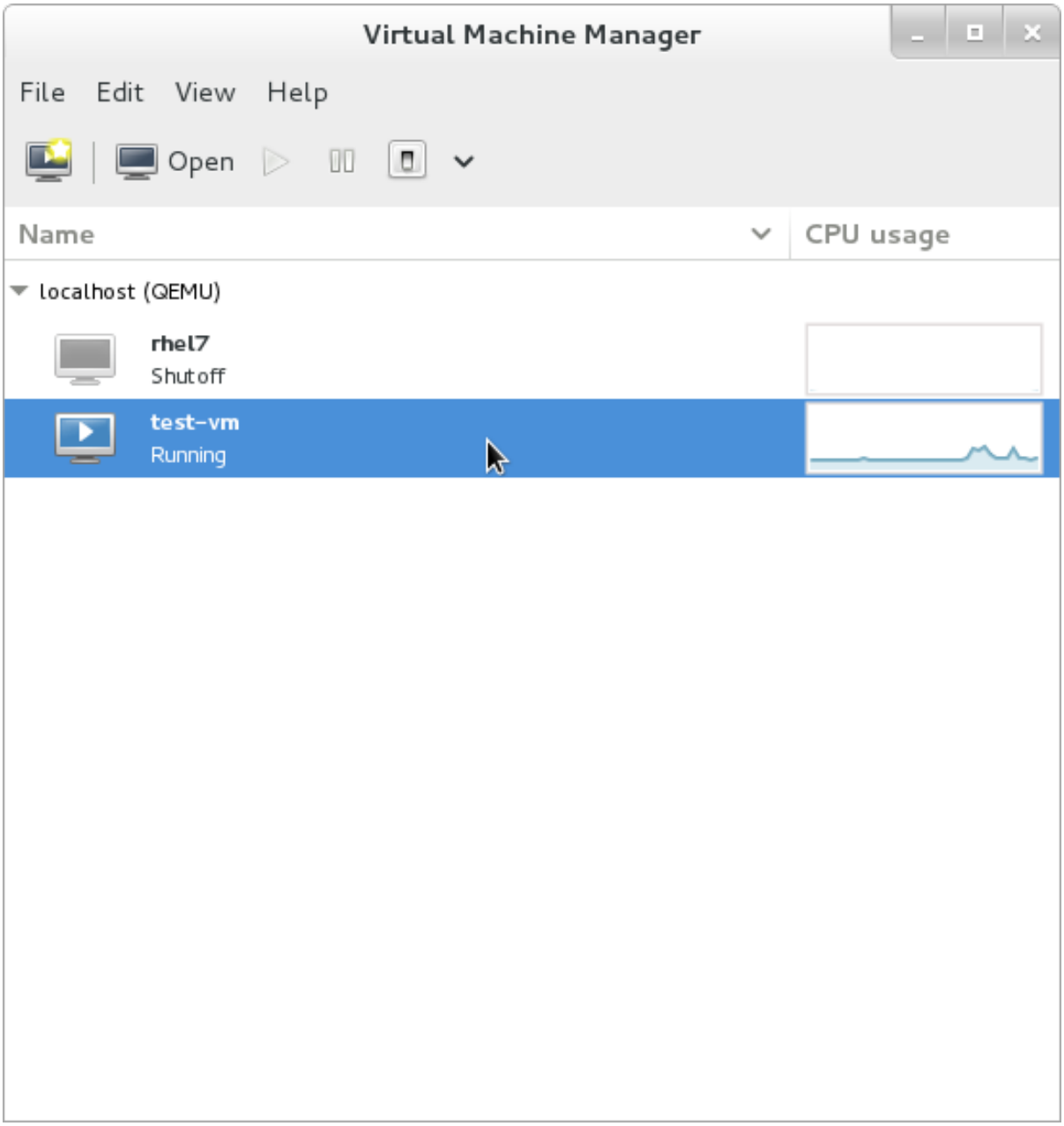


図2.1 表示する仮想マシンの選択

2. 仮想マシンマネージャーの **編集** メニューから **仮想マシンの詳細** を選択します。

   仮想マシンの詳細ウィンドウを開く時にコンソールが表示される場合があります。その場合には、**表示** をクリックしてから **詳細** を選択します。デフォルトでは最初に 概要 ウィンドウが開きます。

3. 左側のナビゲーションペインから **性能 (Performance)** を選択します。

   性能 (Performance) ビューでは、CPU およびメモリーの使用率ならびにディスクおよびネットワークの I/O など、ゲストのパフォーマンスに関する要約が表示されます。

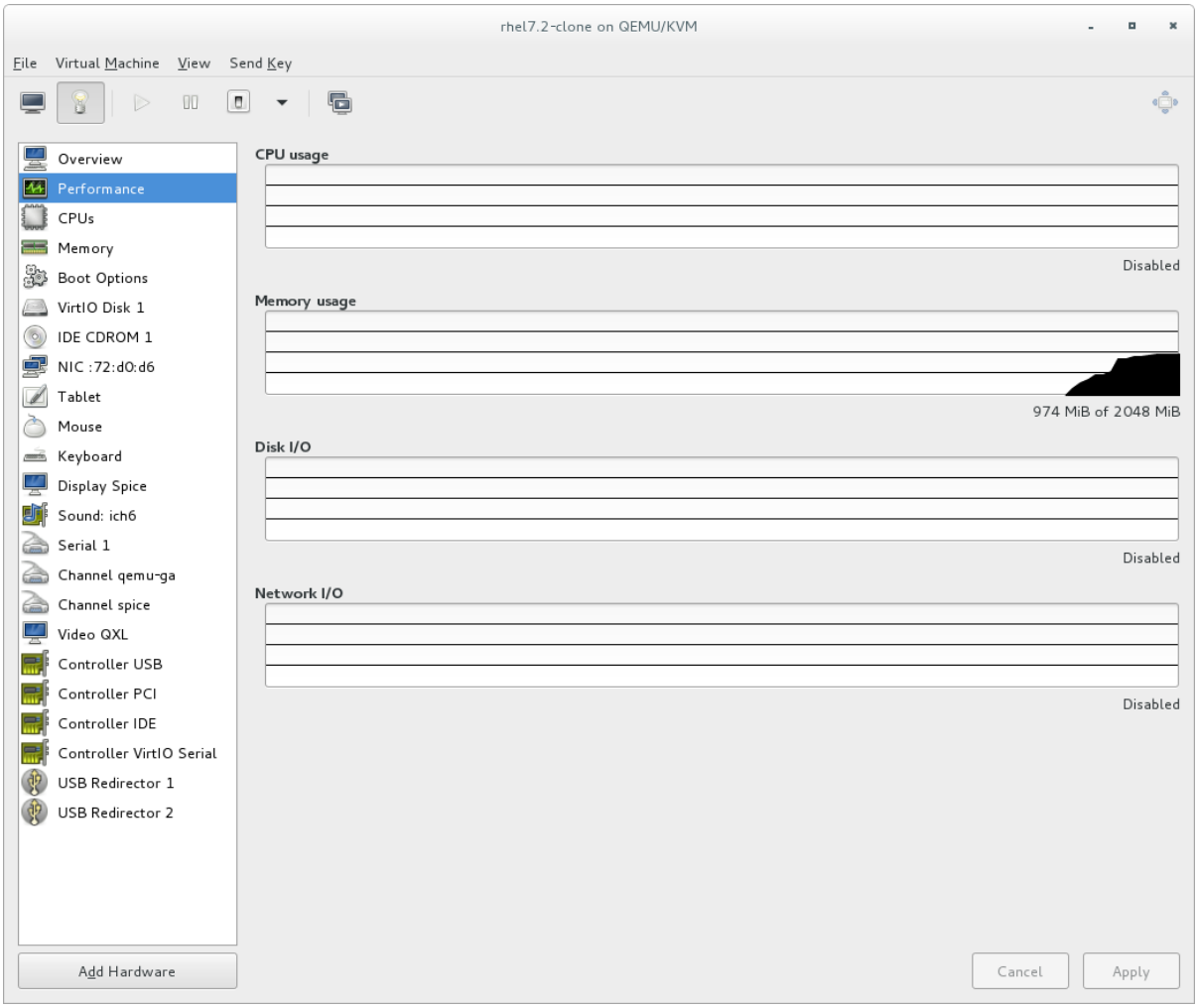


図2.2 ゲストのパフォーマンス詳細の表示

2.3.2. パフォーマンスの監視

パフォーマンスの監視設定は、**virt-manager** の設定ウィンドウで修正することができます。

パフォーマンス監視の設定

1. **編集** メニューから **設定** を選択します。

   設定 ウィンドウが表示されます。

2. **ポーリング** タブで、秒単位の時間または統計ポーリングオプションを指定します。

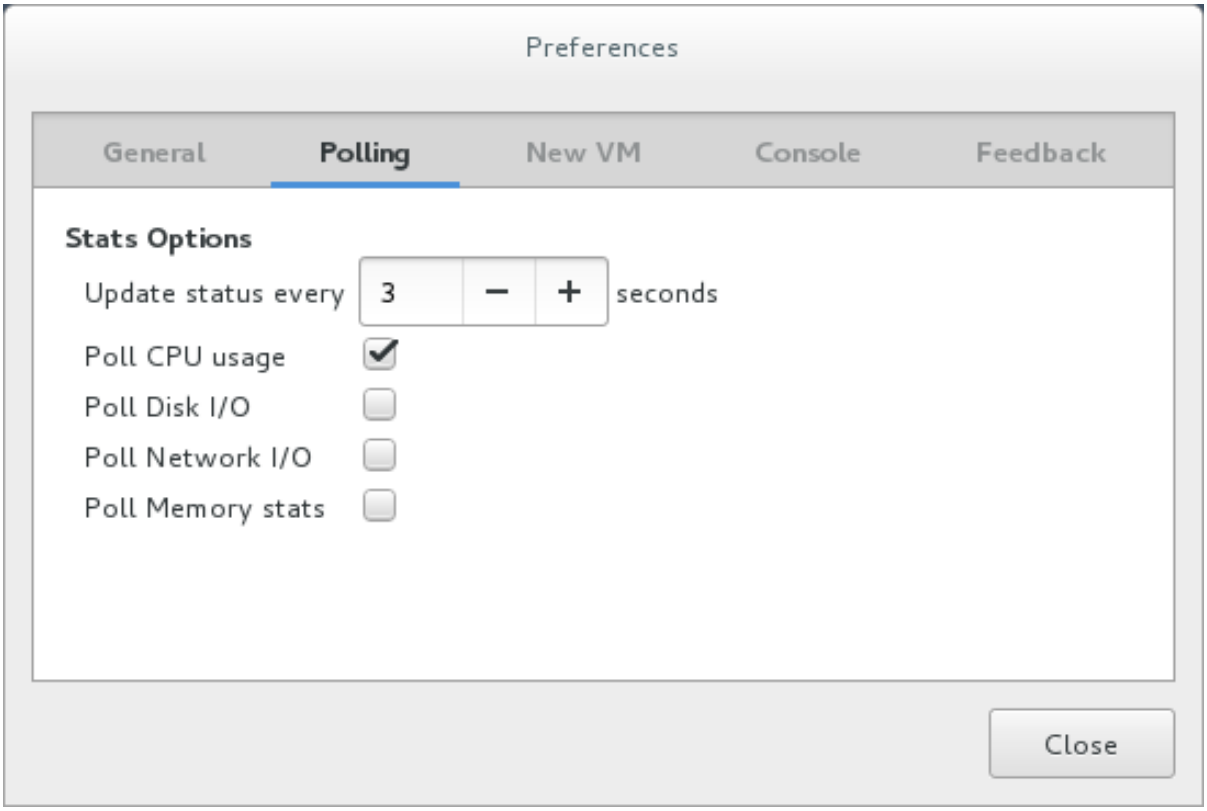


図2.3 パフォーマンス監視の設定

2.3.3. ゲストの CPU 使用率の表示

システム上の全ゲストの CPU 使用率を表示するには、以下を実行します。

1. **表示** メニューから **グラフ** に続いて **仮想マシン CPU 使用率 (Guest CPU Usage)** のチェックボックスを選択します。
2. 仮想マシンマネージャーがシステム上の全仮想マシンの CPU 使用率グラフを表示します。

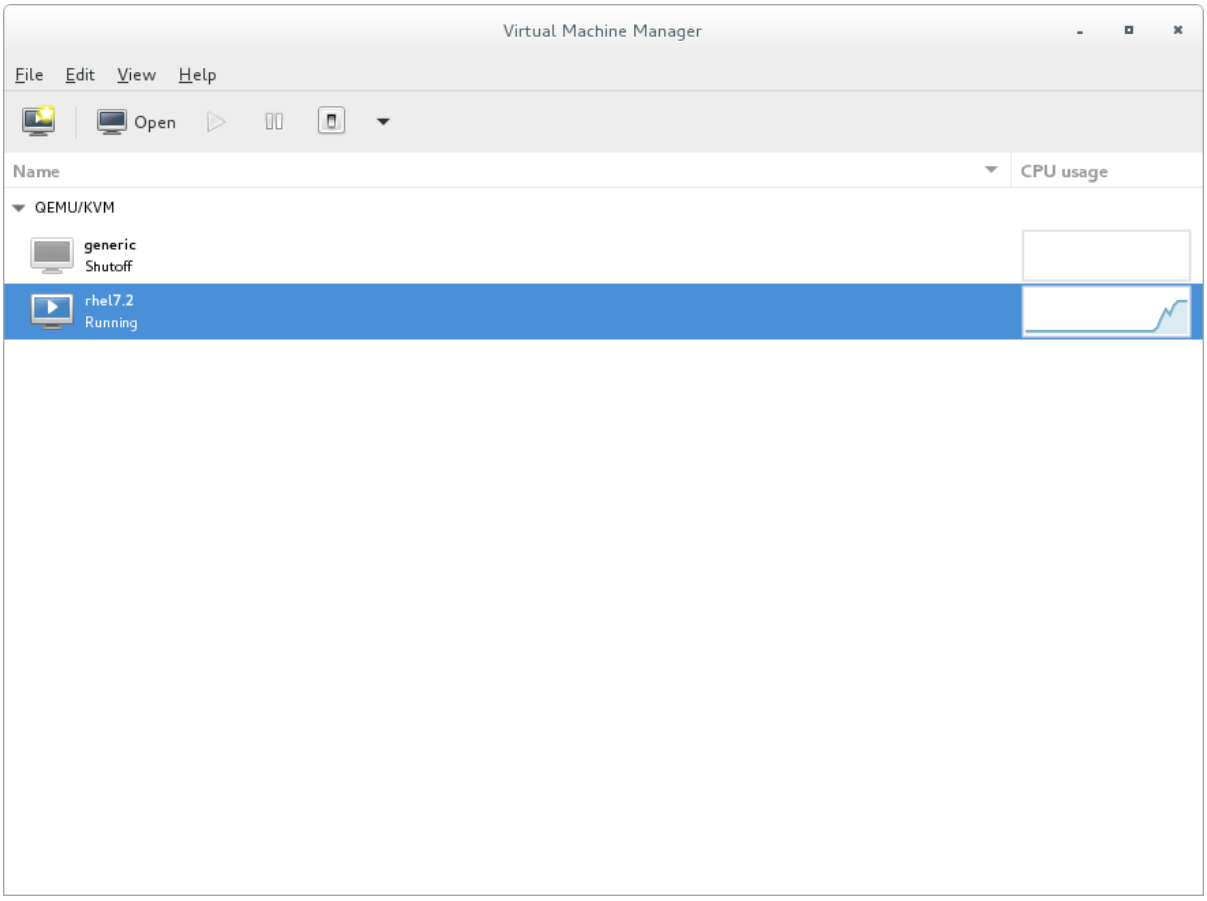


図2.4 ゲストの CPU 使用率グラフ

2.3.4. ホストの CPU 使用率の表示

システム上の全ホストの CPU 使用率を表示するには、以下を実行します。

1. **表示** メニューから **グラフ** に続いて **ホスト CPU 使用率 (Host CPU Usage)** のチェックボックスを選択します。
2. 仮想マシンマネージャーがシステム上のホストの CPU 使用率グラフを表示します。

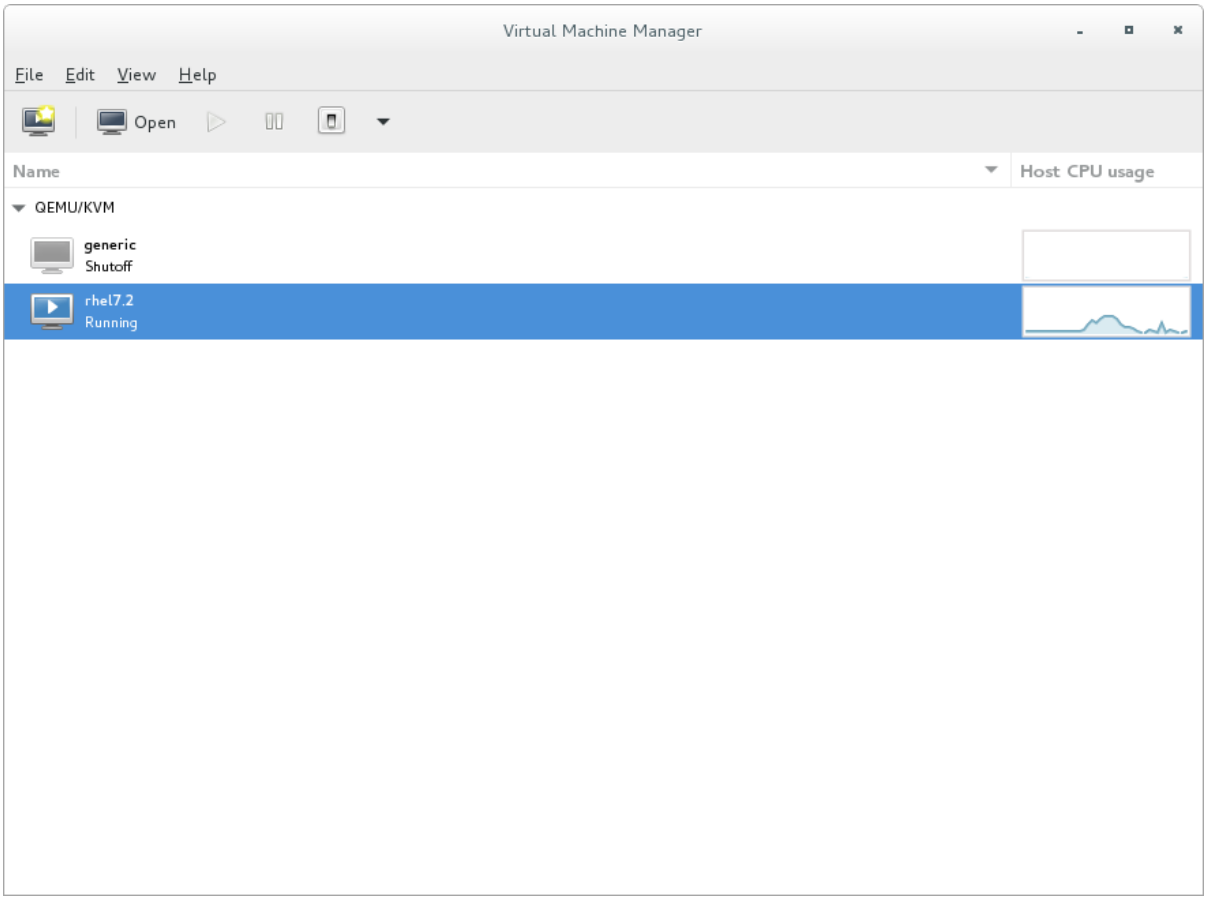


図2.5 ホストの CPU 使用率グラフ

2.3.5. ディスク I/O の表示

システム上の全仮想マシンのディスク I/O を表示するには、以下を実行します。

1. ディスク I/O の統計値収集の設定が有効になっていることを確認します。**編集** メニューから **設定** を選択し、**ポーリング** タブをクリックします。
2. **ディスク I/O の取得** のチェックボックスを選択します。

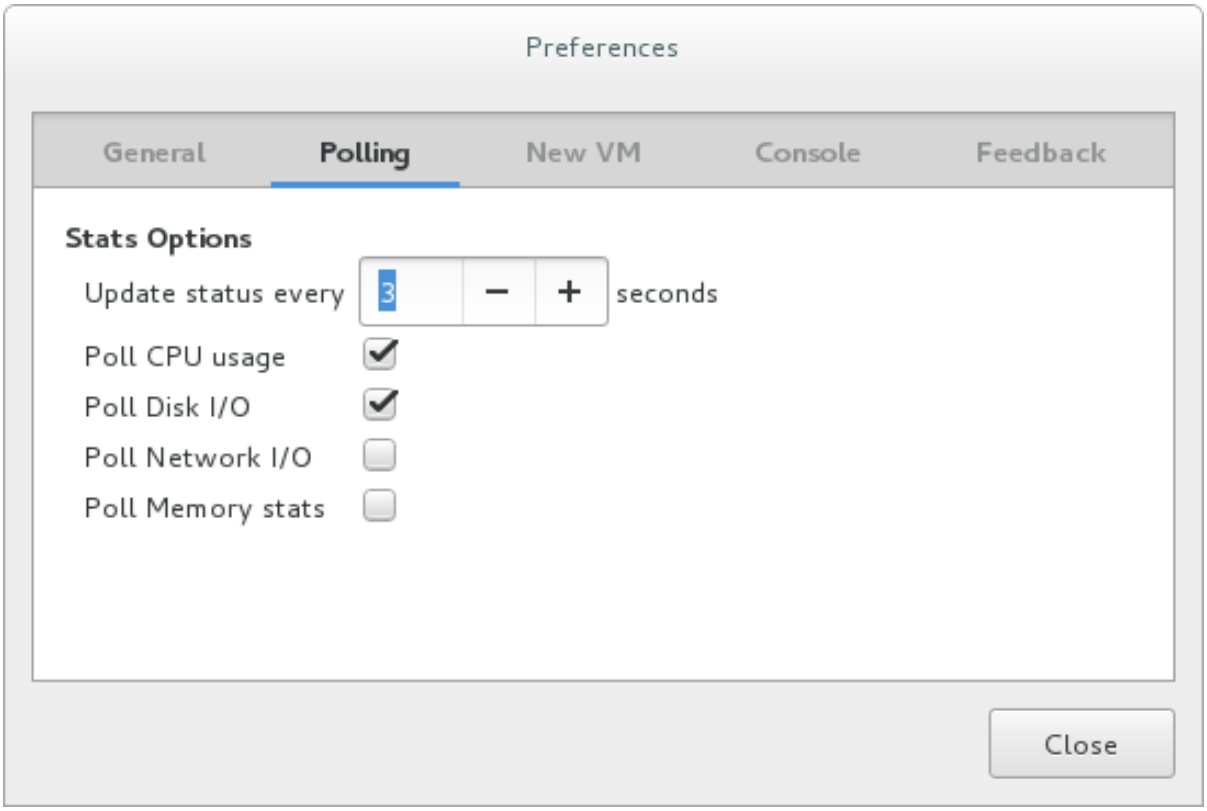


図2.6 ディスク I/O の有効化

3. ディスク I/O の表示を有効にするには、**表示** メニューから **グラフ** に続いて **ディスク I/O** のチェックボックスを選択します。

4. 仮想マシンマネージャーがシステム上の全仮想マシンのディスク I/O のグラフを表示します。

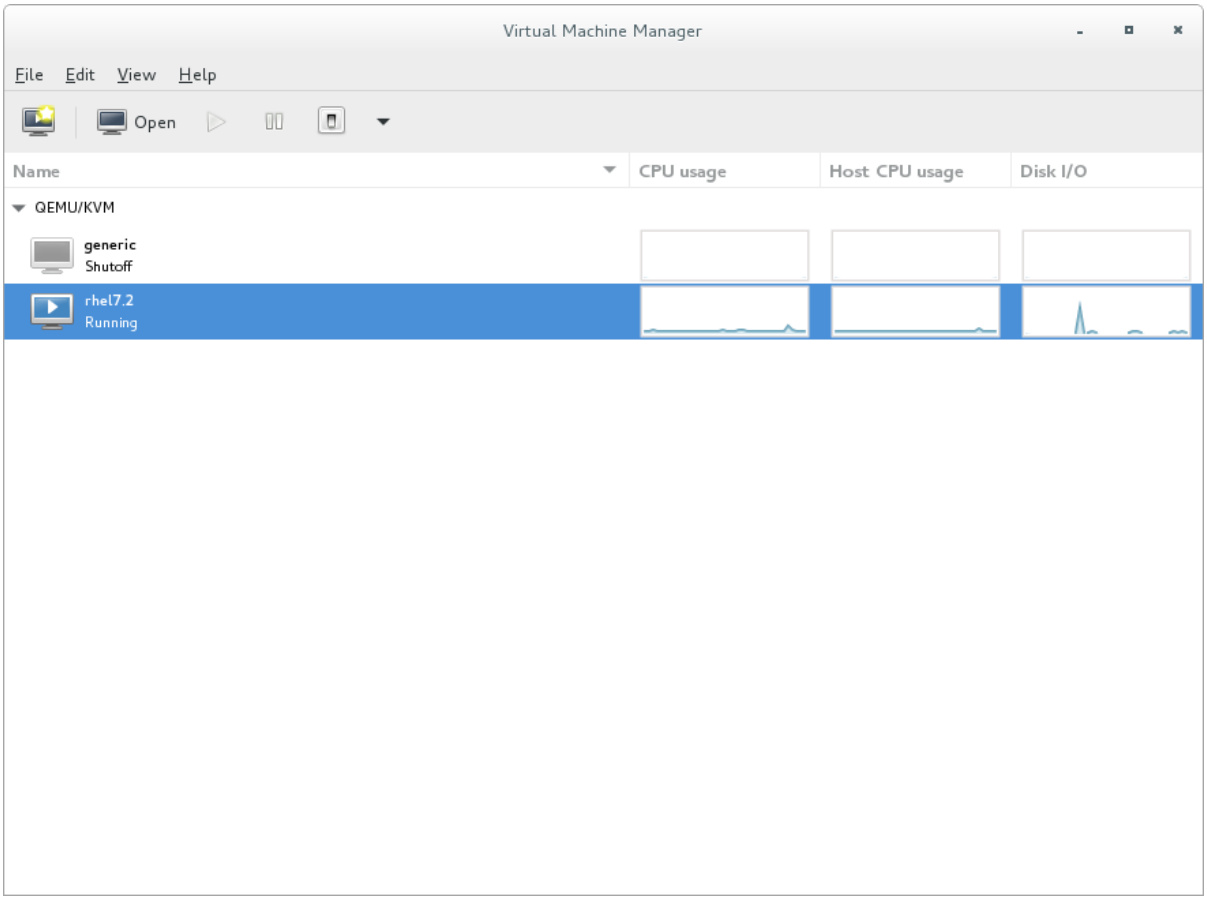


図2.7 ディスク I/O の表示

2.3.6. ネットワーク I/O の表示

システム上の全仮想マシンのネットワーク I/O を表示するには、以下を実行します。

1. ネットワーク I/O の統計値収集の設定が有効になっていることを確認します。これを実行するには、**編集** メニューから **設定** を選択し、**ポーリング** タブをクリックします。
2. **ネットワーク I/O の取得 (Poll Network I/O)** のチェックボックスを選択します。

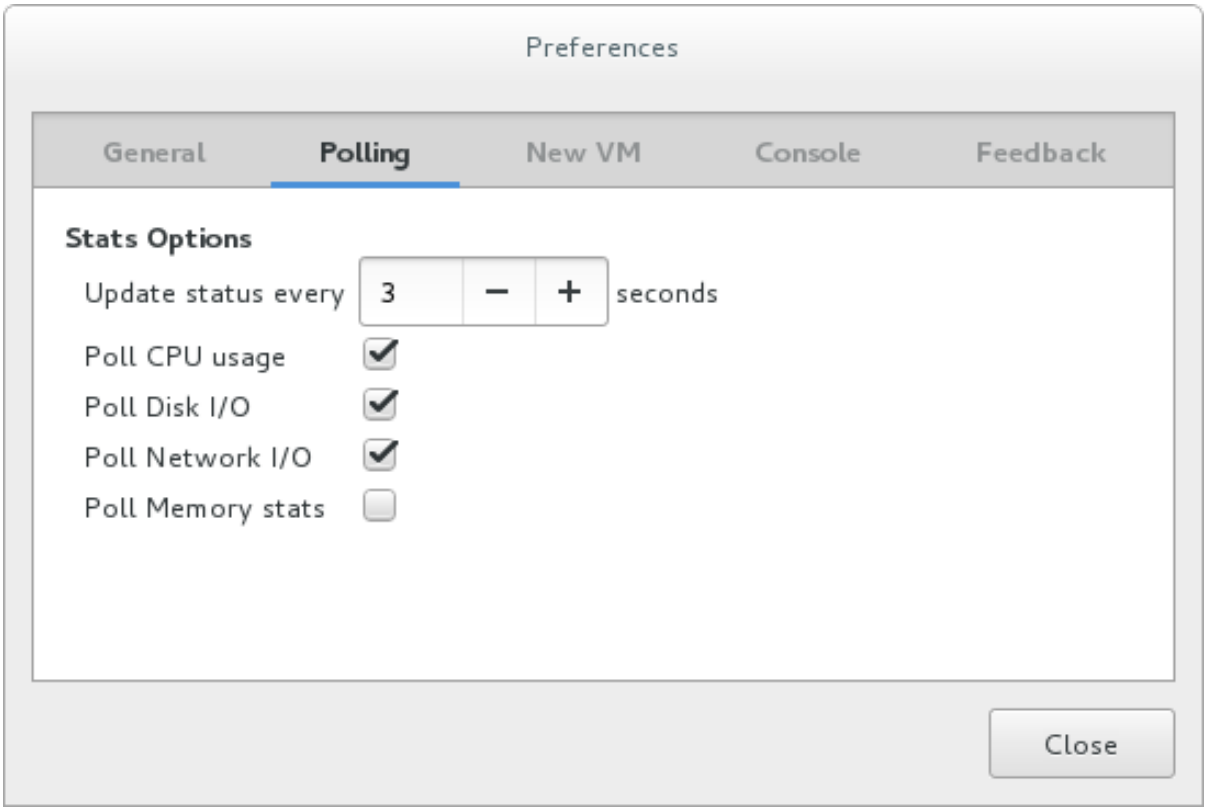


図2.8 ネットワーク I/O の有効化

3. ネットワーク I/O の統計値を表示するには、**表示** メニューから **グラフ** に続いて **ネットワーク I/O** のチェックボックスを選択します。

4. 仮想マシンマネージャーがシステム上の全仮想マシンのネットワーク I/O のグラフを表示します。

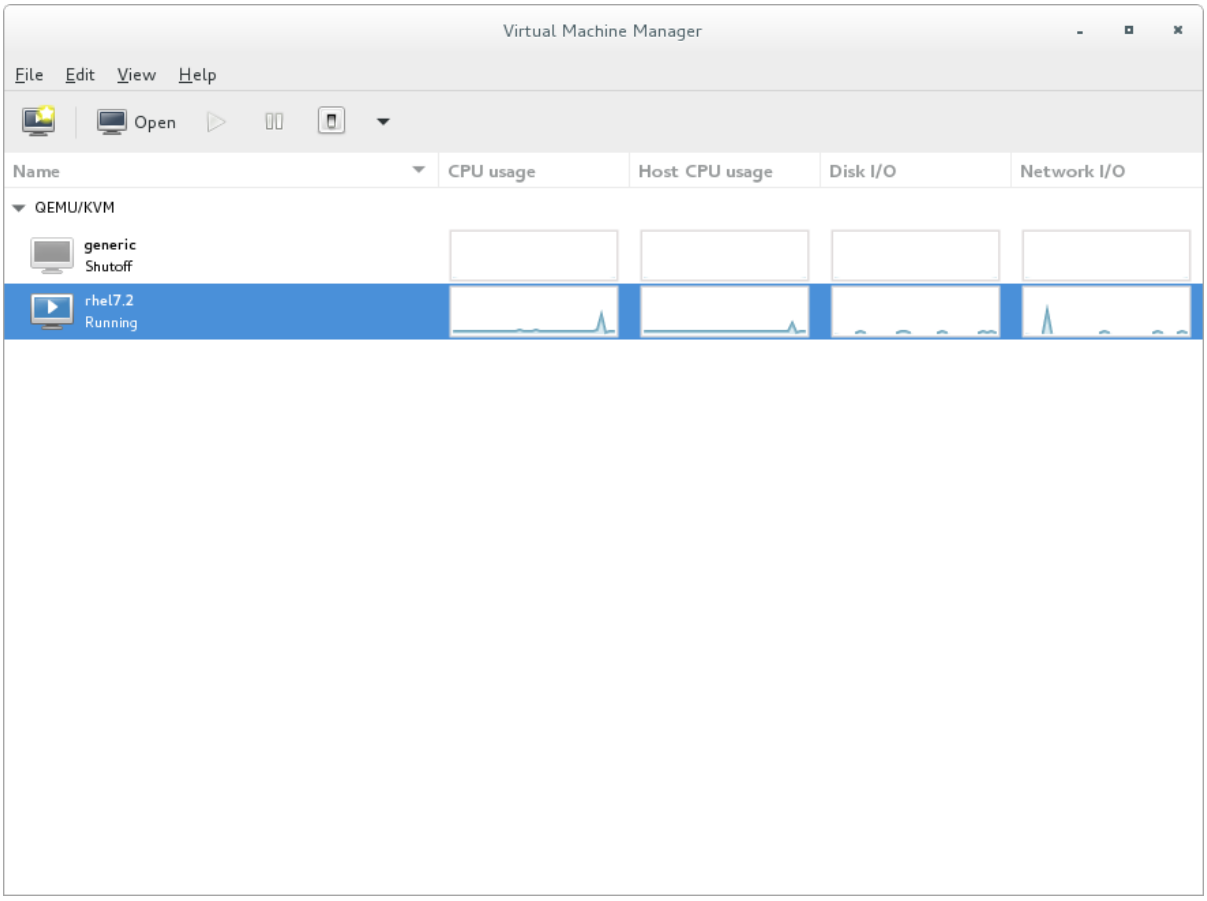


図2.9 ネットワーク I/O の表示

2.3.7. メモリー使用率の表示

システム上のすべての仮想マシンのメモリー使用率を表示するには、以下を実行します。

1. ネットワーク I/O の統計値収集の設定が有効になっていることを確認します。**編集** メニューから **設定** を選択し、**ポーリング** タブをクリックします。

2. **メモリーの統計を取得する (Poll Memory stats)** チェックボックスを選択します。

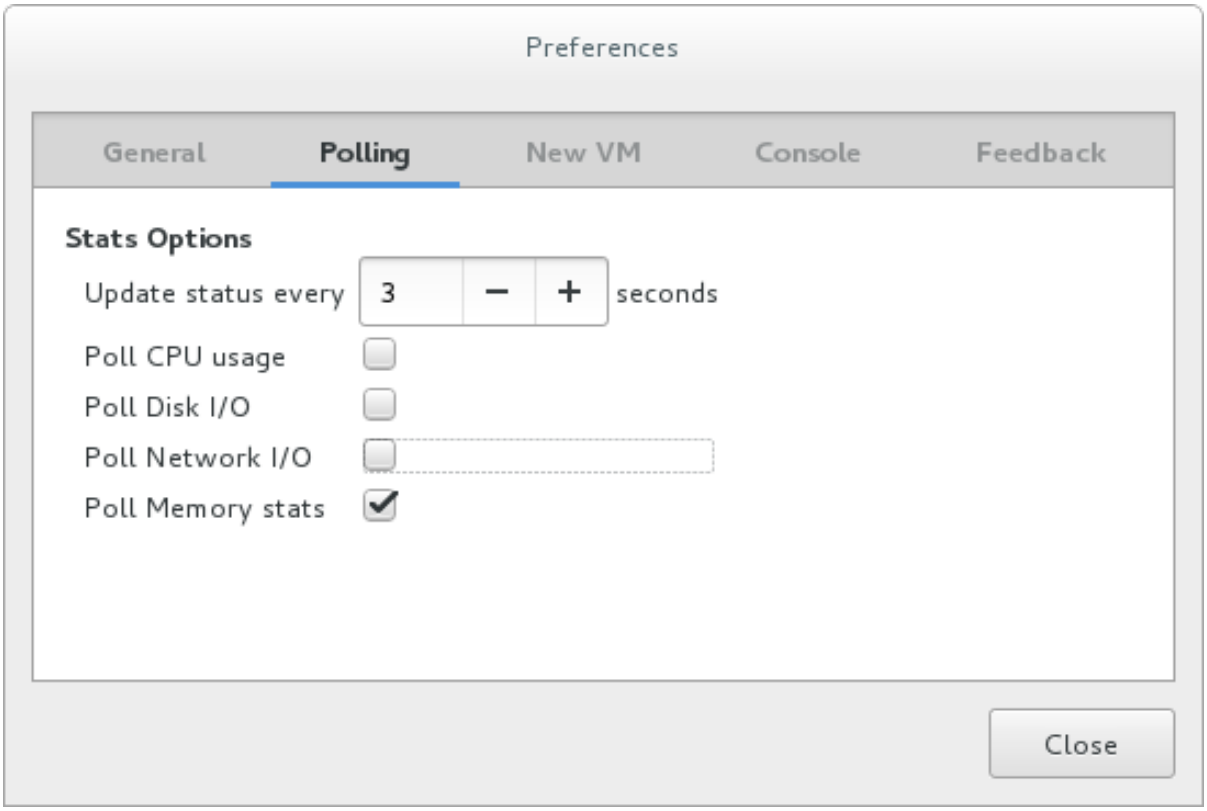


図2.10 メモリー使用率の有効化

3. メモリー使用率を表示するには、**表示** メニューから **グラフ** に続いて **メモリーの使用率** チェックボックスを選択します。

4. 仮想マシンマネージャーがシステム上のすべての仮想マシンについてのメモリー使用率 (メガバイト単位) を一覧表示します。

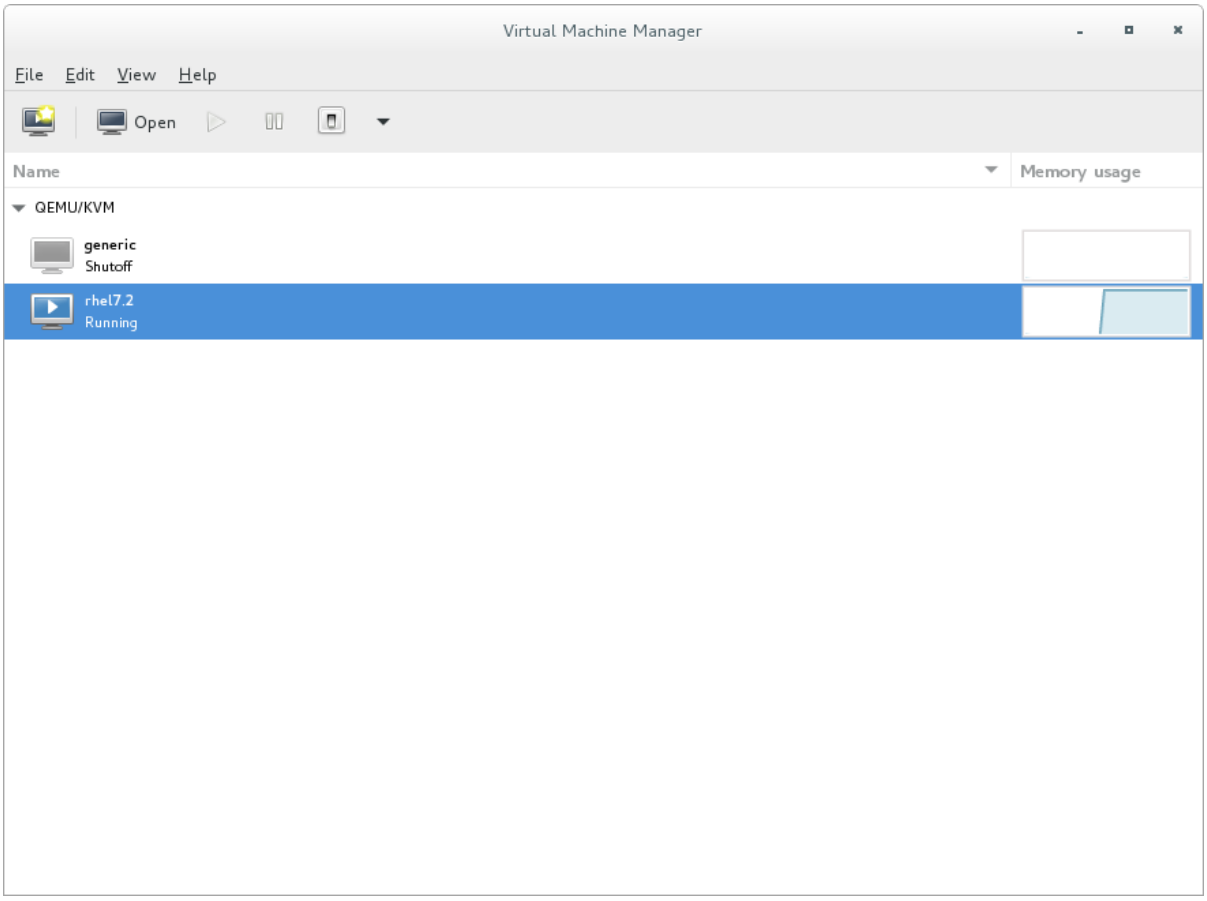


図2.11 メモリー使用率の表示

第3章 VIRT-MANAGER を使用した仮想化パフォーマンスの最適化

本章では、ゲスト仮想マシンの管理を目的としたデスクトップツールとなる virt-manager で利用可能なパフォーマンスチューニング関連のオプションについて扱います。

3.1. オペレーティングシステムの詳細とデバイス

3.1.1. ゲスト仮想マシンの詳細の指定

virt-manager ツールは、新規のゲスト仮想マシンに選択するオペレーティングシステムの種類やバージョンによって異なるプロファイルを提供します。ゲストを作成する際は、できるだけ詳細な情報を入力するようにしてください。特定タイプのゲストに利用できる機能を有効にすることでパフォーマンスを向上させることができます。

virt-manager ツールの以下のサンプルとなるスクリーンショットを参照してください。新規のゲストの仮想マシンを作成する場合は、必ずお使いになる **OS の種類 (OS type)** および **バージョン (Version)** を指定してください。

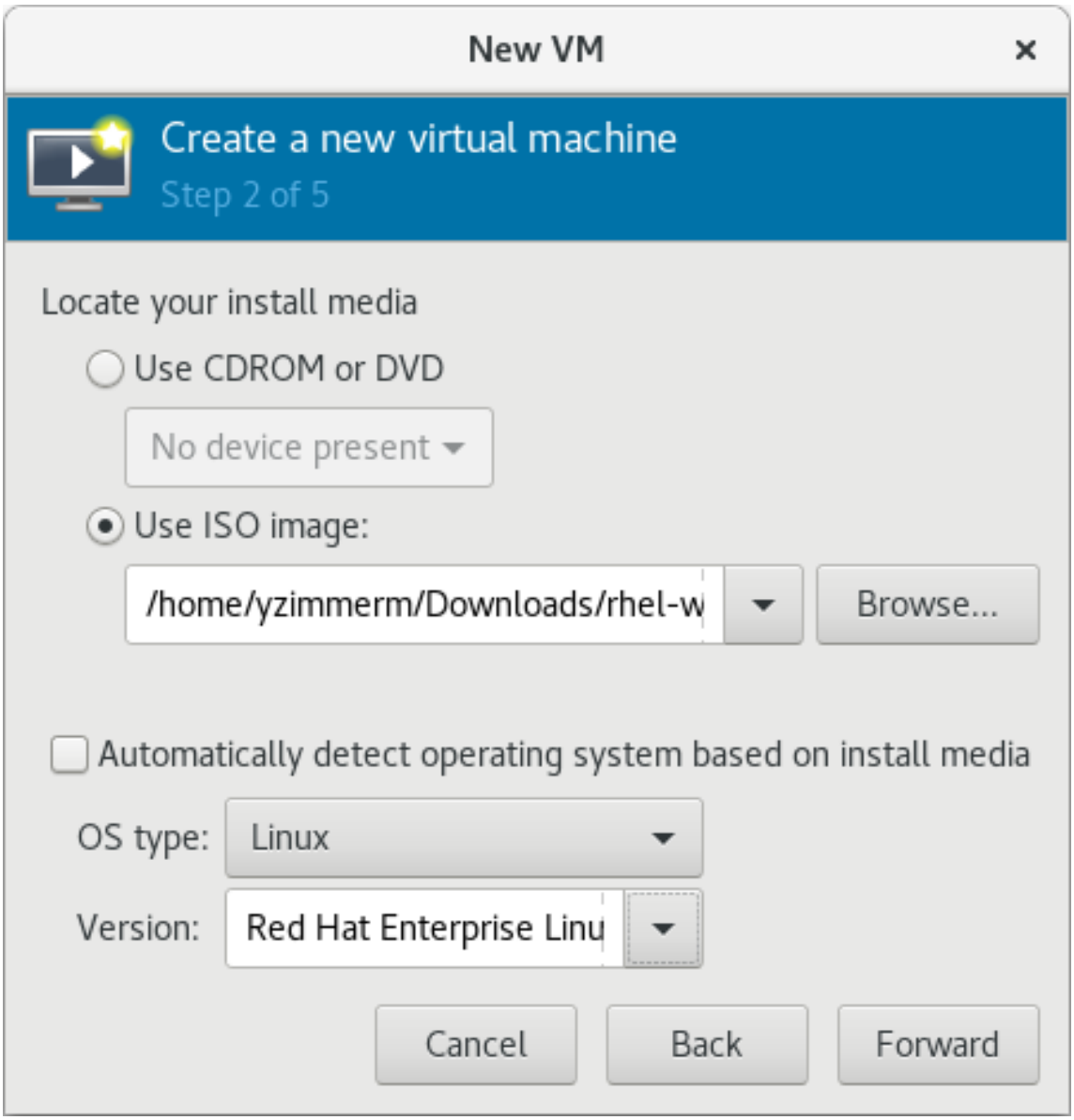


図3.1 OS の種類とバージョンを指定する

3.1.2. 未使用のデバイスの削除

未使用や不要なデバイスを削除するとパフォーマンスが向上する場合があります。たとえば、Web サーバーとして稼働させることを目的としたゲストの場合、オーディオ機能や接続タブレットなどが必要となることはほとんどありません。

virt-manager ツールの以下のサンプルとなるスクリーンショットを参照してください。不要なデバイスを削除するには、**削除 (Remove)** ボタンをクリックします。

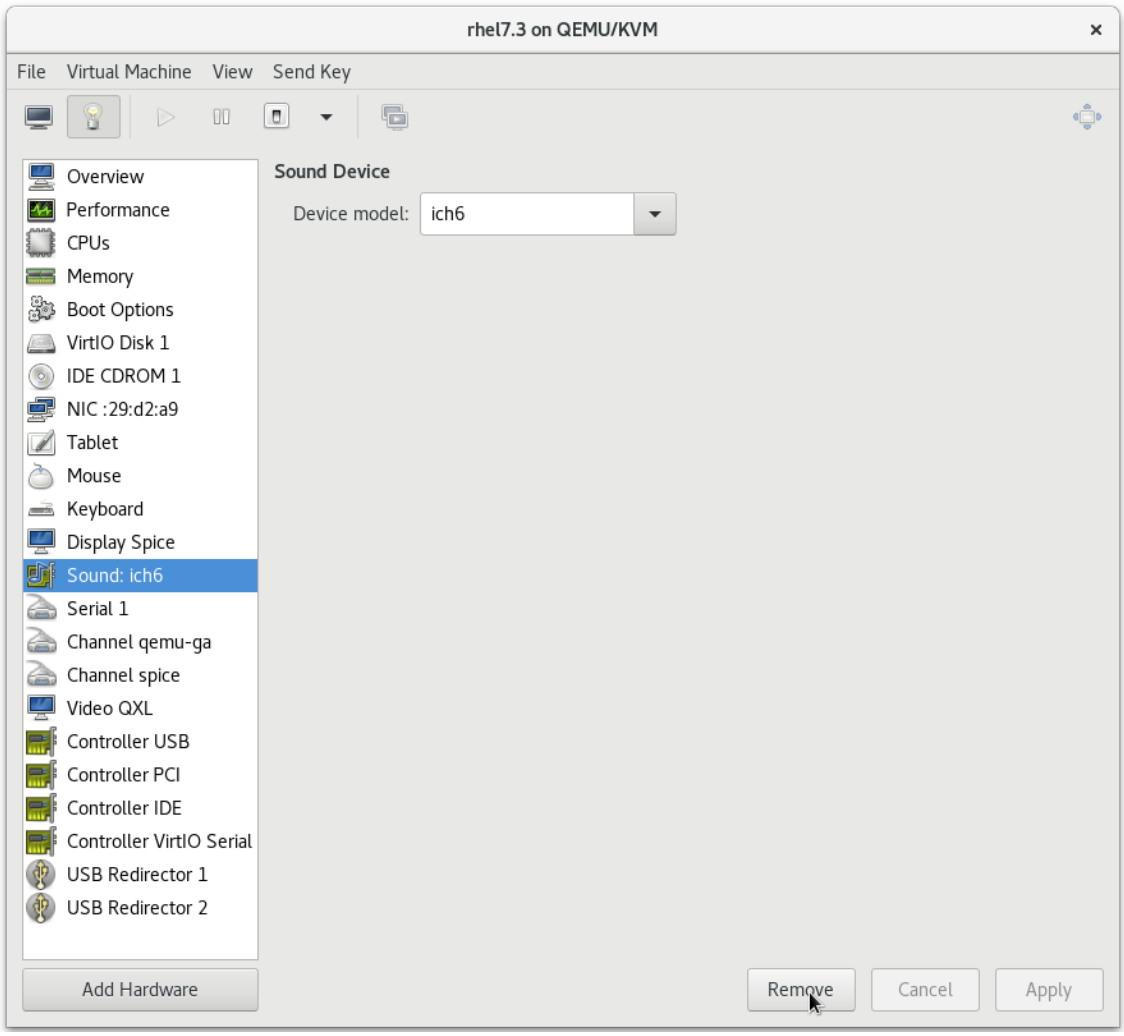


図3.2 未使用のデバイスを削除する

3.2. CPU パフォーマンスのオプション

ゲスト仮想マシンに設定できる CPU 関連のオプションがいくつかあります。パフォーマンスに大きな影響を及ぼす場合があるので、正しく設定してください。ゲストに対して利用できる CPU オプションを以下に示します。このセクションの後半では、これらのオプションがもたらす影響について説明します。

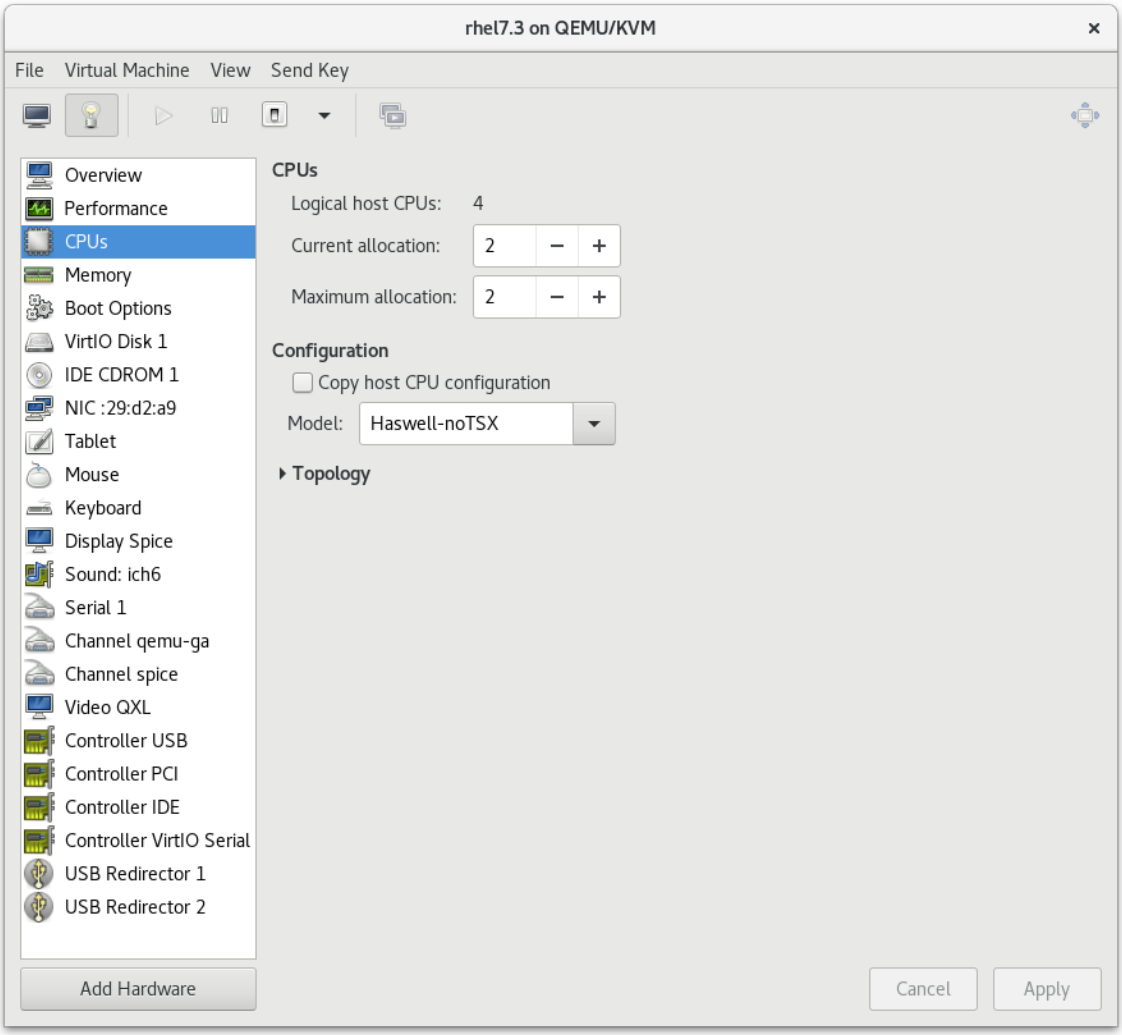


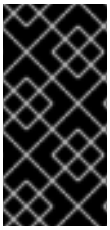
図3.3 CPU パフォーマンスのオプション

3.2.1. オプション: 使用できる CPU

このオプションでゲストが使用できる仮想 CPU (vCPU) の数を調整します。ホストで使用可能な CPU の数を超えて割り当てると (**オーバーコミット** と呼ばれる)、以下のような警告が表示されます。

図3.4 CPU のオーバーコミット

システム上の全ゲストの vCPU の合計がシステム上のホスト CPU の数より大きい場合に CPU はオーバーコミットされます。vCPU の合計数がホスト CPU の数より大きい場合に CPU を 1 つまたは複数のゲストでオーバーコミットする可能性があります。



重要

メモリーのオーバーコミットと同様に、CPU のオーバーコミットはパフォーマンスに悪影響をもたらす場合があります。たとえば、ゲストのワークロードが大きい場合や予想できない場合がこれに該当します。オーバーコミットの詳細は、『仮想化の導入および管理ガイド』を参照してください。

3.2.2. オプション: CPU 構成

このオプションで目的の CPU モデルに応じた CPU 構成タイプを選択します。**ホスト CPU の設定をコピーする (Copy host CPU configuration)** チェックボックスをクリックして、物理ホストの CPU モデルと構成を検出して適用します。あるいは、一覧から、使用できるオプションから選択します。CPU 構成を選択すると、それに応じた CPU 機能と指示が表示され、**CPU 機能 (CPU Features)** の一覧で個別に有効/無効にできます。

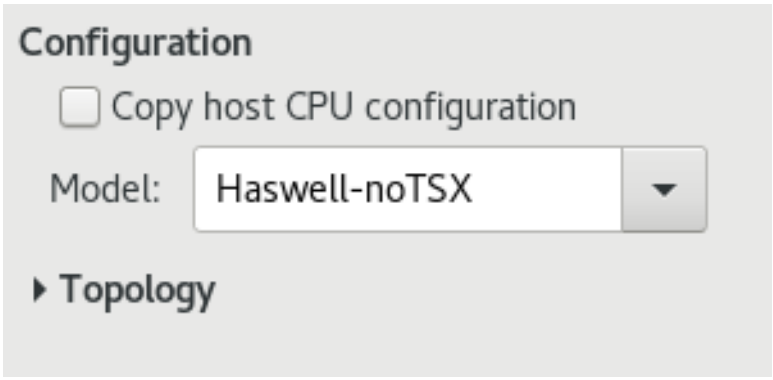
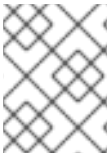


図3.5 CPU 構成のオプション

注記

ホストの CPU 構成は、手動で選択するよりもコピーすることをお勧めします。



注記

別の方法として、ホストマシン上で **virsh capabilities** コマンドを実行し、CPU タイプおよび NUMA 機能を含むシステムの仮想化機能を表示します。

3.2.3. オプション: CPU トポロジー

このオプションでゲスト仮想マシンの仮想 CPU に特定の CPU トポロジーを適用します (ソケット、コア、スレッドなど)。

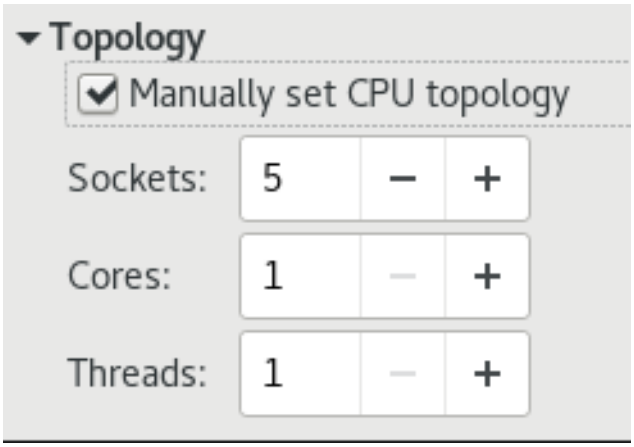


図3.6 CPU トポロジーのオプション

注記

環境によっては要件が異なることもありますが、通常はソケット数を選択する場合に、コア、スレッドのいずれも 1 つのみを指定すると、最善のパフォーマンスを実現できます。

3.3. 仮想ディスクパフォーマンスのオプション

インストール時にゲスト仮想マシンで利用できる仮想ディスク関連のオプションがいくつかあります。この設定により、パフォーマンスに影響を与えることがあります。以下のイメージはゲストで利用できる仮想ディスクのオプションを示しています。

キャッシュモード、IO モード、および IO チューニングは、**virt-manager** の **Virtual Disk** セクションで設定できます。以下の画像で示されるように **Performance options** の下のフィールドにこれらのパラメーターを設定します。

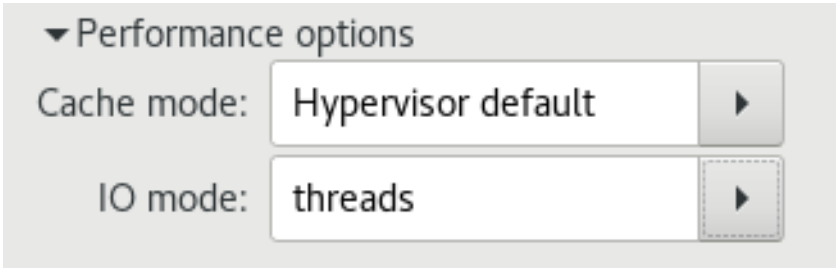
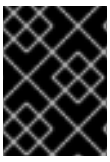


図3.7 仮想ディスクパフォーマンスのオプション



重要

virt-manager で仮想ディスクパフォーマンスのオプションを設定する場合、設定を有効にするために仮想マシンを再起動する必要があります。

これらの設定の説明およびゲスト XML 設定でこれらの設定を編集する方法については、「キャッシュ」および「I/O モード」を参照してください。

第4章 TUNED と TUNED-ADM

本章では **tuned** デーモンを使って仮想化環境下のシステム設定をチューニングする方法について説明しています。

tuned は、CPU 集約型タスク、またはストレージおよびネットワークのスループット応答性などの特定のワークロードの特定に対して、Red Hat Enterprise Linux を適用させるチューニングプロファイル配信メカニズムです。これにより、数々の特定のユースケースでパフォーマンスを向上し、電力消費量を抑えることができます。プロファイルを編集したり、新規のプロファイルを作成することで、環境に適したパフォーマンスソリューションを作り出すことができます。

tuned の一部として提供される仮想化関連のプロファイルには、以下のものがあります。

virtual-guest

throughput-performance プロファイルに基づき、***virtual-guest*** は仮想メモリーの スワップアウト処理頻度も軽減します。

virtual-guest プロファイルは Red Hat Enterprise Linux 7 ゲスト仮想マシンを作成する際に自動的に選択されます。これは仮想マシンに推奨されるプロファイルです。

このプロファイルは Red Hat Enterprise Linux 6.3 以降で利用できますが、仮想マシンのインストール時に手動で選択する必要があります。

virtual-host

throughput-performance プロファイルに基づき、***virtual-host*** もダーティーページのより集中的なライトバックを可能にします。このプロファイルは、KVM と Red Hat Virtualization (RHV) ホストの両方を含む仮想化ホストのプロファイルとして推奨されます。

Red Hat Enterprise Linux 7 インストールでは、デフォルトで tuned パッケージがインストールされ、**tuned** サービスが有効にされます。

利用可能な全プロファイルを一覧表示して、現在アクティブなプロファイルを特定するには、以下を実行します。

```
# tuned-adm list
Available profiles:
- balanced
- desktop
- latency-performance
- network-latency
- network-throughput
- powersave
- sap
- throughput-performance
- virtual-guest
- virtual-host
Current active profile: throughput-performance
```

さらに、一連のチューニングパラメーターをカプセル化するためにカスタム **tuned** プロファイルを作成することもできます。カスタム **tuned** プロファイルの作成方法には、**tuned.conf** の man ページを参照してください。

現在アクティブなプロファイルだけを表示する場合は、以下を実行します。

```
tuned-adm active
```

別のプロファイルに切り替える場合は、以下を実行します。

```
tuned-adm profile profile_name
```

たとえば、**virtual-host** プロファイルに切り替えるには、以下を実行します。

```
tuned-adm profile virtual-host
```

重要

Red Hat Enterprise Linux 7.1 以降で tuned プロファイルを設定したら、再起動後に適用するように設定したプロファイルで **tuned** が有効となるようにします。

```
# systemctl enable tuned
```

手動で設定されたパラメーターを使用するために **tuned** を無効にする方が望ましい場合があります。現在のセッションですべてのチューニングを無効にするには、以下を実行します。

```
# tuned-adm off
```

tuned を永続的に無効にし、すべての変更を元に戻す場合は、以下のコマンドを実行します。

```
# tuned-adm off; systemctl disable tuned
```

注記

tuned の詳細は、『Red Hat Enterprise Linux 7 パフォーマンスチューニングガイド』を参照してください。

第5章 ネットワーク

本章では、仮想化環境におけるネットワークの最適化について説明します。

5.1. ネットワークチューニングのヒント

- 単一ネットワークでのトラフィック混雑を避けるため複数のネットワークを使用します。たとえば、管理専用のネットワーク、バックアップ専用のネットワーク、ライブマイグレーション専用のネットワークなどを設けます。
- Red Hat では、同じネットワークセグメントで複数のインターフェースを使用しないことを推奨しています。***arp_filter*** を使用することで ARP 変動を防ぐことができます。ARP 変動はホストおよびゲストのいずれでも発生する可能性のある望ましくない状況で、マシンが複数のネットワークインターフェースから ARP 要求に応答してしまう原因となります。この設定を永続化するには、**echo 1 > /proc/sys/net/ipv4/conf/all/arp_filter** を実行するか、または **/etc/sysctl.conf** を編集してください。

注記

ARP 変動の詳細は、http://linux-ip.net/html/ether-arp.html#ether-arp-flux を参照してください。

5.2. VIRTIO と VHOST_NET

以下の図は、Virtio および vhost_net アーキテクチャーにおけるカーネルの位置付けを示しています。

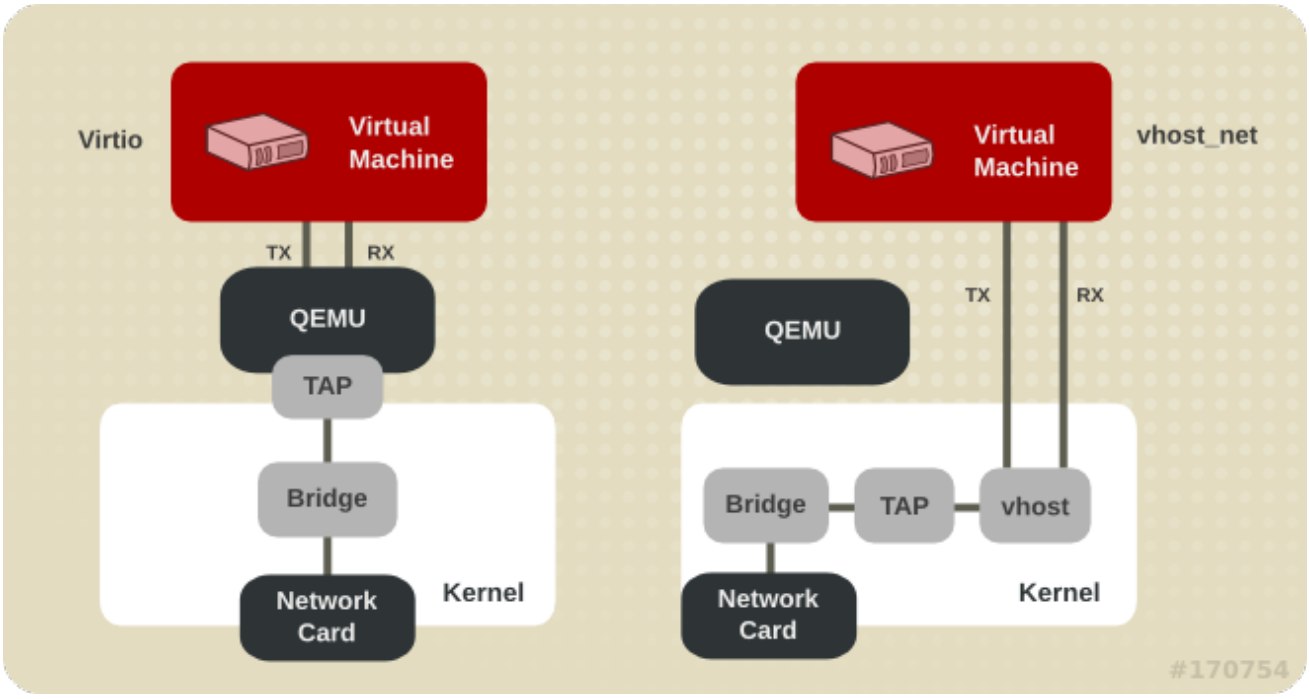


図5.1 Virtio および vhost_net アーキテクチャー

vhost_net は virtio ドライバーの一部をユーザー領域からカーネルに移動します。これによりコピー操作が減り、待ち時間と CPU 使用量が低減します。

5.3. デバイスの割り当てと SR-IOV

デバイス割り当てと SR-IOV の構造におけるカーネルの位置付けを示します。

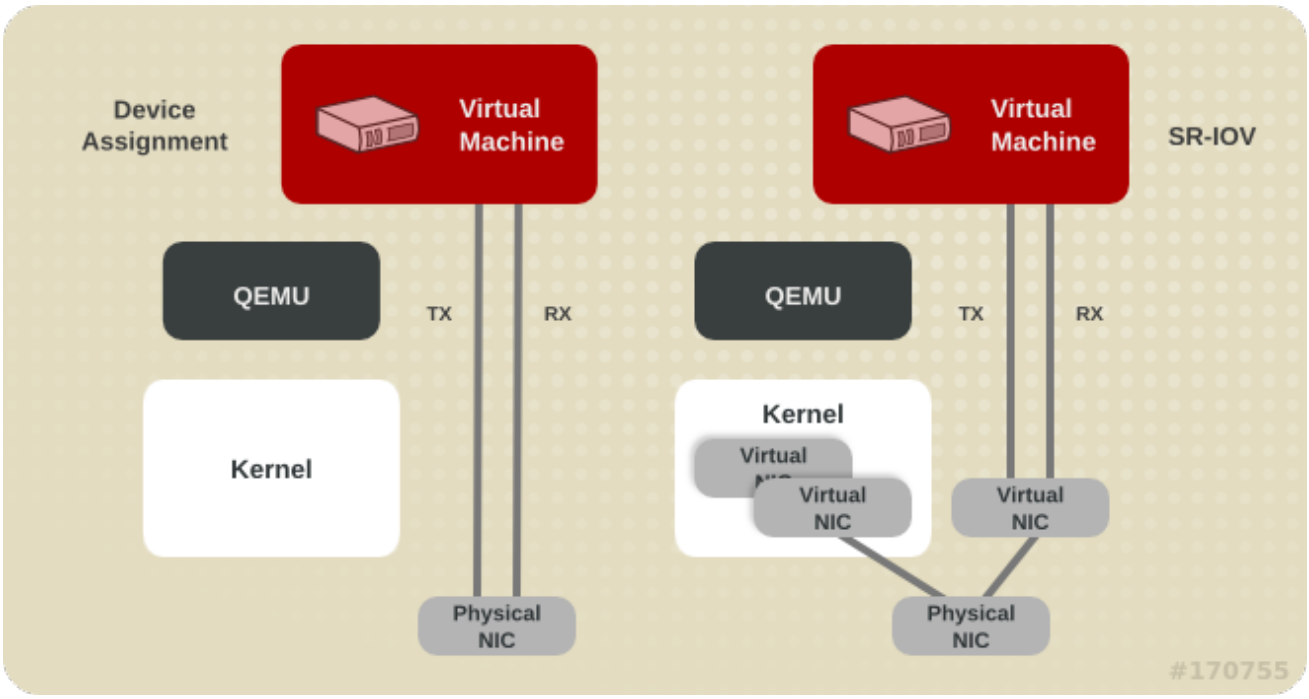
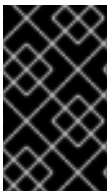


図5.2 デバイスの割り当てと SR-IOV

デバイス割り当てにより、デバイス全体がゲストから見える状態になります。SR-IOV では、NIC やシステムボードなどドライバーやハードウェアのサポートを必要とします。複数の仮想デバイスを作成することが可能なため、それぞれ別々のゲストに渡すことができます。ゲスト内に製造元固有のドライバーが必要になりますが、SR-IOV はいずれのネットワークオプションでも最小の待ち時間を実現します。

5.4. ネットワークチューニングのヒント

ここでは、仮想化環境におけるネットワークのパフォーマンスチューニングの方法について説明します。



重要

以下の機能は、Red Hat Enterprise Linux 7 ハイパーバイザーおよび仮想マシンでサポートされていますが、Red Hat Enterprise Linux 6.6 以降を実行する仮想マシンでもサポートされています。

5.4.1. ブリッジのゼロコピー送信

ゼロコピー送信モードは、大規模なパケットサイズで有効です。これは通常、ゲストネットワークと外部ネットワーク間で大規模なパケットを送信する場合に、スループットに影響を与えることなく、最大 15% までホスト CPU のオーバーヘッドを削減します。

これはゲスト間、ゲスト対ホスト、または小規模なパケットワークロードのパフォーマンスには影響を与えません。

ブリッジのゼロコピー送信は、Red Hat Enterprise Linux 7 仮想マシンで完全にサポートされていますが、デフォルトでは無効になっています。ゼロコピー送信モードを有効にするには、vhost_net モジュールの ***experimental_zcopytx*** カーネルモジュールパラメーターを 1 に設定します。『仮想化の導入および管理ガイド』を参照してください。

注記

通常、サービス拒否および情報漏洩攻撃による脅威の軽減策として追加のデータコピーが送信時に作成されます。ゼロコピー送信を有効にすると、この脅威の軽減策が無効にされます。

パフォーマンスの低下が観察される場合やホスト CPU の利用率が問題にならない場合は、***experimental_zcopytx*** を 0 に設定することでゼロコピー送信モードを無効にできます。

5.4.2. マルチキュー virtio-net

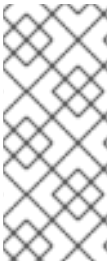
マルチキュー virtio-net は、1 度に複数の virtqueue ペアでパケットを送信できるようにすることで、vCPU の数の増大と共にネットワークパフォーマンスを増大させるアプローチを提供します。

現在のハイエンドサーバーにはより多くのプロセッサーがあり、それらで実行されるゲストの vCPU の数も増加しています。単一キューの virtio-net ではネットワークパフォーマンスが vCPU の数の増加と共に増加しないため、ゲスト内のプロトコルスタックのスケールは制限されています。また virtio-net には TX および RX キューが 1 つだけあるため、ゲストはパケットを並行して送信したり、取得したりすることができません。

マルチキューサポートは、並列パケット処理を許可することにより、これらのボトルネックを取り除きます。

マルチキュー virtio-net は、以下の場合に最高のパフォーマンスを提供します。

- トラフィックパケットのサイズが比較的大きい。
- ゲスト間、ゲストとホスト間、またはゲストと外部システム間でトラフィックが実行されている状態で、ゲストが多くの接続で同時にアクティブである。
- キューの数は vCPU の数に等しい。これは、特定のキューを特定の vCPU に対してプライベートにするため、マルチキューサポートが RX 割り込みの親和性および TX キュー選択を最適化するためです。



注記

現在、multi-queue virtio-net を設定すると、発信トラフィックのパフォーマンスに影響を及ぼす可能性があります。たとえば、TCP (Transmission Control Protocol) ストリームで 1,500 バイト以下のパケットを送信すると、この問題が発生する場合があります。詳細は、「About the impact of virtio-net multi-queue on tx performance」 を参照してください。

5.4.2.1. マルチキュー virtio-net の設定

マルチキュー virtio-net を使用するには、以下をゲスト XML 設定に追加してゲストのサポートを有効にします (カーネルはマルチキュータップデバイスについて最高 256 のキューをサポートするため、ここでは *N* の値を 1 から 256 に設定します)。

```
<interface type='network'>
    <source network='default'/>
    <model type='virtio'/>
    <driver name='vhost' queues='N'/>
</interface>
```

ゲスト内で *N* virtio-net キューを指定して仮想マシンを実行する場合、以下のコマンドを使ってマルチキューサポートを有効にします (ここで、*M* の値は 1 から *N* になります)。

```
# ethtool -L eth0 combined M
```

5.5. ネットワークパケットのバッチ処理

転送パスが長い状況では、パケットをバッチ処理してからカーネルに送信することでキャッシュが有効に活用される場合があります。

バッチ処理することのできるパケットの最大数を設定するには、以下のコマンドを実行します。ここで、*N* はバッチ処理するパケットの最大数です。

```
# ethtool -C $tap rx-frames N
```

type='bridge' または type='network' のインターフェースにおいて **tun**/**tap** の rx バッチ処理をサポートするには、ドメイン XML ファイルに以下のようなスニペットを追加します。

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet0'/>
      <coalesce>
        <rx>
          <frames max='7'/>
        </rx>
      </coalesce>
  </interface>
</devices>
```

第6章 I/O スケジューリング

Red Hat Enterprise Linux 7 が仮想化 ホスト の場合、およびそれが仮想化 ゲスト である場合は、ディスクのパフォーマンスを向上させるために、入力/出力 (I/O) スケジューラーを使用できます。

6.1. RED HAT ENTERPRISE LINUX を仮想化ホストとして使用する場合の I/O スケジューリング

Red Hat Enterprise Linux 7 を仮想ゲストのホストとして使用する場合、通常はデフォルトの **deadline** スケジューラーが最良です。このスケジューラーは、ほとんどすべてのワークロードに対して適切に機能します。

I/O スループットを最大化することが、ゲストのワークロードの I/O レイテンシーを最小化よりも重要である場合は、代わりに **cfq** スケジューラーを使用すると役に立つ可能性があります。

6.2. RED HAT ENTERPRISE LINUX を仮想化ゲストとして使用する場合の I/O スケジューリング

Red Hat Enterprise Linux ゲスト仮想マシンにおいて、I/O スケジューリングを使用することができます。この場合、ゲストを実行中のハイパーバイザーに制限はありません。考慮すべきメリットおよびデメリットを、一覧にして以下に示します。

- Red Hat Enterprise Linux ゲストでは、多くの場合、**noop** スケジューラーを使用することで大きな恩恵を受けます。このスケジューラーは、ハイパーバイザーに I/O を送る前に、ゲストオペレーティングシステムからの小規模なリクエストを大きなリクエストにまとめます。これにより、ハイパーバイザーが I/O リクエストをより効果的に処理できるようになり、ゲストの I/O パフォーマンスが大幅に向上します。

- ワークロードの I/O およびストレージデバイスのマウント状況によっては、**deadline** などのスケジューラーの方が役に立つ場合があります。Red Hat は、どのスケジューラーで最高のパフォーマンスが得られるかを確認するために、パフォーマンステストを行うことを推奨します。

- iSCSI、SR-IOV、または物理デバイスのパススルーによりストレージにアクセスするゲストの場合は、**noop** スケジューラーを使用するべきではありません。これらの手段では、ホストはベースとなる物理デバイスに対する I/O 要求を最適化することができません。

注記

仮想化環境では、ホストおよびゲストの両レイヤーで I/O をスケジューリングしても役に立たない場合があります。複数のゲストがファイルシステムのストレージや、ホストオペレーティングシステムの管理するブロックデバイスを使用する状況では、ホストがより効率的に I/O をスケジューリングできる場合があります。これは、ホストは全ゲストからの要求を把握し、ストレージの物理レイアウト (ゲストの仮想ストレージとはリニアに対応していない場合があります) を理解しているためです。

スケジューラーのチューニングは、必ず通常の運用状態でテストする必要があります。総合的なベンチマークでは、仮想環境の共有リソースを使用するシステムのパフォーマンスを正確に比較することができないためです。

6.2.1. Red Hat Enterprise Linux 7 における I/O スケジューラーの設定

Red Hat Enterprise Linux 7 システムで使用されるデフォルトのスケジューラーは **deadline** ですが、Red Hat Enterprise Linux 7 のゲストマシンでは、以下のようにしてスケジューラーを **noop** に変更すると役に立つ可能性があります。

1. **/etc/default/grub** ファイルでは、**GRUB_CMDLINE_LINUX** 行の **elevator=deadline** 文字列を **elevator=noop** に変更します。**elevator=** 文字列がない場合は、行末に **elevator=noop** を追加します。

   変更に成功すると、以下により、**/etc/default/grub** ファイルが表示されます。

   ```
   # cat /etc/default/grub
   [...]
   GRUB_CMDLINE_LINUX="crashkernel=auto rd.lvm.lv=vg00/lvroot rhgb quiet
   elevator=noop"
   [...]
   ```

2. **/boot/grub2/grub.cfg** ファイルを再構築します。

   - BIOS ベースのシステムの場合は、以下のコマンドを実行します。

     ```
     # grub2-mkconfig -o /boot/grub2/grub.cfg
     ```

   - UEFI ベースのマシンの場合は、以下のコマンドを実行します。

     ```
     # grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
     ```

第7章 ブロック I/O

本章では、仮想化環境における I/O の最適化について説明します。

7.1. ブロック I/O チューニング

virsh blkiotune コマンドを使うと、管理者はゲスト仮想マシンのブロック I/O パラメーターをゲスト XML 設定の **<blkio>** 要素に手動で設定したり、表示したりすることができます。

仮想マシンの現在の **<blkio>** パラメーターを表示するには、以下を実行します。

```
# virsh blkiotune virtual_machine
```

仮想マシンの **<blkio>** パラメーターを設定するには、**virsh blkiotune** コマンドを使用し、ご使用の環境に応じて値を置き換えます。

```
# virsh blkiotune virtual_machine [--weight number] [--device-weights string] [--config] [--live] [--current]
```

パラメーターには以下が含まれます。

weight

I/O ウェイト値 (100 から 1000 の範囲内)。

デバイスの I/O ウェイトを上げると、I/O 帯域幅の優先度が高まるため、より多くのホストリソースが提供されます。同様に、デバイスの加重を下げると、ホストリソースの消費量が少なくなります。

device-weights

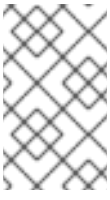
/path/to/device,weight,/path/to/device,weight 形式で 1 つ以上の device/weight のペアを一覧表示する単一文字列です。それぞれのウェイトは 100-1000 の範囲内にあるか、またはデバイスごとの一覧からデバイスを削除するには値は 0 である必要があります。文字列に一覧表示されているデバイスのみが変更されます。他のデバイスの既存のデバイスごとのウェイトは変更されません。

config

次回の起動時に変更を有効にするには、**--config** オプションを追加します。

live

実行中の仮想マシンに変更を適用するには、**--live** オプションを追加します。



注記

--live オプションは、ハイパーバイザーがこのアクションをサポートするように要求します。すべてのハイパーバイザーがメモリー上限のライブの変更を許可する訳ではありません。

current

変更を現在の仮想マシンに適用するには、**--current** オプションを追加します。

たとえば、以下は、*liftbrul* 仮想マシンの **/dev/sda** デバイスの重みを 500 に変更します。

```
# virsh blkiotune liftbrul --device-weights /dev/sda, 500
```

注記

virsh blkiotune コマンド使用の詳細については、**virsh help blkiotune** コマンドを使用してください。

7.2. キャッシュ

キャッシュオプションは、ゲストのインストール時に **virt-manager** で設定することも、ゲスト XML 設定を編集して既存のゲスト仮想マシン上で設定することもできます。

表7.1 キャッシュオプション

キャッシュオプション	説明
Cache=none	ゲストからの I/O はホストではキャッシュされませんが、ライトバックディスクキャッシュに保持することができます。大規模な I/O 要件のあるゲストにはこのオプションを使用します。一般的に、このオプションは移行に対応する最適で唯一のオプションとなります。
Cache=writethrough	ゲストからの I/O はホストにはキャッシュされ、物理的な媒体に書き込むことができます。このモードを使用すると速度の低下が生じ、ゲスト数の規模が一定以上になると問題が発生する傾向にあります。I/O 要求が低い少数のゲストへの使用に適しています。ゲスト移行の必要性がなく、ライトバックキャッシュに対応していないゲストに推奨されるモードです(Red Hat Enterprise Linux 5.5 以前)。
Cache=writeback	ゲストからの I/O がホストでキャッシュされます。
Cache=directsync	***writethrough*** と似ていますが、ゲストからの I/O はホストページのキャッシュをバイパスします。
Cache=unsafe	ホストはすべてのディスク I/O をキャッシュする可能性がありますが、ゲストからの同期要求は無視されます。
Cache=default	キャッシュモードが指定されない場合、システムのデフォルト設定が選択されます。

virt-manager では、キャッシュモードは **Virtual Disk** 以下で指定できます。キャッシュモードを変更するために **virt-manager** を使用する方法についての詳細は、「仮想ディスクパフォーマンスのオプション」を参照してください。

ゲスト XML 内でキャッシュモードを設定するには、**driver** タグ内で **cache** 設定を編集し、キャッシュオプションを指定します。たとえば、キャッシュを ***writeback*** として設定するには、以下を実行します。

```
<disk type='file' device='disk'>
        <driver name='qemu' type='raw' cache='writeback'/>
```

7.3. I/O モード

I/O オプションは、ゲストのインストール時に **virt-manager** で設定することも、ゲスト XML 設定を編集して既存のゲスト仮想マシン上で設定することもできます。

表7.2 I/O モードのオプション

I/O モードのオプション	説明
IO=native	Red Hat Virtualization (RHV) 環境のデフォルトです。このモードは直接 I/O オプションが設定されたカーネルの非同期 I/O を参照します。
IO=threads	デフォルトは、ホストユーザーモードベースのスレッドです。
IO=default	Red Hat Enterprise Linux 7 におけるデフォルトはスレッドモードです。

virt-manager では、I/O モードは **Virtual Disk** 以下で指定できます。I/O モードを変更するために **virt-manager** を使用する方法についての詳細は、「仮想ディスクパフォーマンスのオプション」を参照してください。

ゲスト XML 内で I/O モードを設定するには、**driver** タグ内で **io** 設定を編集し、***native***、***threads***、または ***default*** を指定します。たとえば、I/O モードを ***threads*** に設定するには、以下を実行します。

```
<disk type='file' device='disk'>
        <driver name='qemu' type='raw' io='threads'/>
```

7.4. ブロック I/O のチューニング方法

ここでは、仮想化環境におけるブロック I/O パフォーマンス調整の詳細の方法について説明します。

7.4.1. ディスク I/O スロットリング

複数の仮想マシンが同時に実行される場合、それらは過剰なディスク I/O を使用することでシステムのパフォーマンスに影響を与える可能性があります。KVM のディスク I/O スロットリングは、仮想マシンからホストマシンに送られるディスク I/O 要求に制限を設定する機能を提供します。これにより、1 台の仮想マシンが共有リソースを過剰に使用し、他の仮想マシンのパフォーマンスに影響を与えることを防ぐことができます。

ディスク I/O スロットリングは、異なる顧客に属するゲスト仮想マシンが同じホストで実行されている場合や、異なるゲストについてサービスの品質保証が提供されている場合などの各種の状況で役立ちます。ディスク I/O スロットリングは、低速なディスクをシミュレーションするために使用することもできます。

I/O スロットリングは、ゲストに割り当てられた各ブロックデバイスに独自に適用でき、スループットおよび I/O 操作上の制限をサポートします。**virsh blkdeviotune** コマンドを使用して、仮想マシンに I/O 制限を設定します。

```
# virsh blkdeviotune virtual_machine device --parameter limit
```

device には、仮想マシンに割り当てられているディスクデバイスのいずれかの固有のターゲット名 (**<target dev='name'/>**) またはソースファイル (**<source file='name'/>**) を指定します。ディスクデバイス名の一覧については、**virsh domblklist** コマンドを使用します。

オプションのパラメーターには以下が含まれます。

total-bytes-sec

秒あたりのバイト単位の合計スループット制限

read-bytes-sec

秒あたりのバイト単位の読み込みスループット制限

write-bytes-sec

秒あたりのバイト単位の書き込みスループット制限

total-iops-sec

秒あたりの合計 I/O 回数の制限

read-iops-sec

秒あたりの読み込み I/O 回数の制限

write-iops-sec

秒あたりの書き込み I/O 回数の制限

たとえば、**virtual_machine** 上の **vda** を 1 秒あたり 1000 I/O 操作および 1 秒あたり 50 MB スループットにスロットリングするには、以下のコマンドを実行します。

```
# virsh blkdeviotune virtual_machine vda --total-iops-sec 1000 --total-bytes-sec 52428800
```

7.4.2. マルチキュー virtio-scsi

マルチキュー virtio-scsi は、virtio-scsi ドライバーにおける強化されたストレージパフォーマンスとスケーラビリティーを提供します。これにより、他の vCPU に影響を与えることなく、それぞれの仮想 CPU で別個のキューや割り込みを使用できます。

7.4.2.1. マルチキュー virtio-scsi の設定

マルチキュー virtio-scsi は Red Hat Enterprise Linux 7 ではデフォルトで無効にされています。

ゲストでマルチキュー virtio-scsi サポートを有効にするには、以下をゲスト XML 設定に追加します。ここで、*N* は vCPU キューの合計数です。

```
<controller type='scsi' index='0' model='virtio-scsi'>
<driver queues='N' />
 </controller>
```

第8章 メモリー

本章では、仮想化環境におけるメモリーの最適化オプションについて説明しています。

8.1. メモリーチューニングのヒント

仮想化環境でメモリーパフォーマンスを最適化するには、以下を考慮してください。

- 使用する量よりも多くのリソースをゲストに割り当てない。
- (可能な場合) リソースが NUMA ノードに十分にある場合はゲストを単一の NUMA ノードに割り当てます。NUMA の使用についての詳細は、「9章*NUMA*」を参照してください。

8.2. 仮想マシン上でのメモリーチューニング

8.2.1. メモリー監視ツール

メモリーの使用は、ベアメタル環境で使用されるツールを使って仮想マシンで監視することができます。メモリー使用の監視とメモリー関連の問題の診断に役立つツールには以下が含まれます。

- **top**
- **vmstat**
- **numastat**
- **/proc/**

注記

これらのパフォーマンスツールの使用方法は、『Red Hat Enterprise Linux 7 パフォーマンスチューニングガイド』、およびこれらのコマンドの man ページを参照してください。

8.2.2. virsh を使用したメモリーチューニング

ゲスト XML 設定のオプションの **<memtune>** 要素により、管理者はゲスト仮想マシンのメモリー設定を手動で設定することができます。**<memtune>** が省略される場合、デフォルトのメモリー設定が適用されます。

virsh memtune コマンドを使って、仮想マシン内の **<memtune>** 要素にメモリーパラメーターを表示または設定します。ご使用の環境に応じて値を置き換えます。

```
# virsh memtune virtual_machine --parameter size
```

オプションのパラメーターには以下が含まれます。

hard_limit

仮想マシンが使用できる最大メモリーです。この値はキビバイト (1024 バイトのブロック) で表されます。

警告

この値の設定が低すぎると、仮想マシンがカーネルによって kill される可能性があります。

soft_limit

これは、メモリー競合中に強制するメモリーの制限です。この値はキビバイト (1024 バイトのブロック) で表されます。

swap_hard_limit

これは、仮想マシンが使用できる最大メモリーに swap を足したものです。この値はキビバイト (1024 バイトのブロック) で表されます。***swap_hard_limit*** の値は ***hard_limit*** 値よりも大きくなければなりません。

min_guarantee

これは、仮想マシンへの割り当てを保証できる最小メモリーです。この値はキビバイト (1024 バイトのブロック) で表されます。

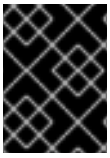
注記

virsh memtune コマンドの使用方法についての詳細は、**# virsh help memtune** を参照してください。

オプションの **<memoryBacking>** 要素には、仮想メモリーページがホストページで保護される方法に影響を与えるいくつかの要素が含まれる場合があります。

locked を設定すると、ホストがゲストに属するメモリーページをスワップアウトすることを回避します。ホストのメモリーで仮想メモリーページをロックするには、以下をゲスト XML に追加します。

```
<memoryBacking>
      <locked/>
</memoryBacking>
```



重要

locked を設定する際、**<memtune>** 要素の ***hard_limit*** には、ゲストに設定された最大メモリーにプロセス自体で消費されたメモリーを足したものを設定する必要があります。

nosharepages を設定すると、ホストがゲスト間で使用される同じメモリーをマージすることを避けられます。ハイパーバイザーに対してゲストの共有ページを無効にするよう指示するには、以下をゲストの XML に追加します。

```
<memoryBacking>
      <nosharepages/>
</memoryBacking>
```

8.2.3. Huge Page および Transparent Huge Page

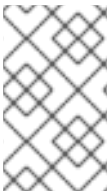
通常、AMD64 および Intel 64 CPU は 4 kB ページ単位でメモリーに対応しますが、*Huge Page* とも言われる 2 MB または 1 GB の大容量ページを使用することも可能です。*TLB (Transaction Lookaside Buffer)* に対して CPU キャッシュの使用を増加させてパフォーマンスを向上させる場合は、Huge Page メモリー対応の KVM のゲストを導入することができます。

Huge Page はカーネルの機能で、Red Hat Enterprise Linux 7 ではデフォルトで有効になっています。とくに大容量のメモリーやメモリー集約型のワークロードでパフォーマンスが大幅に向上します。Red Hat Enterprise Linux 7 では、Huge Page を使用することでページサイズを拡大し、大容量メモリーをより効率的に管理できます。Huge Page の有効性および利便性を改善するために、Red Hat Enterprise Linux 7 はデフォルトで *Transparent Huge Pages* (THP) を使用します。Huge Page および THP の詳細は、『パフォーマンスチューニングガイド』を参照してください。

Red Hat Enterprise Linux 7 システムは、システムの起動時またはランタイム時に割り当てることができる 2 MB および 1 GB の Huge Page をサポートします。複数の Huge Page のページサイズを有効にする方法は、「起動時またはランタイム時におけるゲストの 1 GB Huge Page の有効化」を参照してください。

8.2.3.1. Transparent Huge Page の設定

THP (Transparent huge pages) は、Huge Page の作成、管理、使用のほとんどを自動化する抽象化レイヤーです。デフォルトでは、パフォーマンスに対するシステム設定を自動的に最適化します。



注記

KSM を使用すると、THP (transparent huge page) の発生を低減できるため、THP を有効にする前に KSM を無効にすることが推奨されます。詳細は「KSM の非アクティブ化」を参照してください。

Transparent huge page はデフォルトで有効になります。現在のステータスを確認するには、以下を実行します。

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
```

Transparent Huge Page のデフォルト使用を可能にするには、以下を実行します。

```
# echo always > /sys/kernel/mm/transparent_hugepage/enabled
```

これにより、**/sys/kernel/mm/transparent_hugepage/enabled** が ***always*** に設定されます。

Transparent Huge Page を無効にするには、以下を実行します。

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

Transparent Huge Page サポートによって静的な Huge Page の使用が妨げられることはありません。ただし、静的な Huge Page が使用されない場合には、通常の 4 kb のページサイズではなく Transparent Huge Page が KVM によって使用されます。

8.2.3.2. 静的 Huge Page の設定

huge page の制御を強化することが望ましい場合があります。ゲスト上で静的な huge page を使用するには、**virsh edit** を使用してゲスト XML 設定に以下を追加します。

-

```
<memoryBacking>
      <hugepages/>
</memoryBacking>
```

これは、ホストに対して、デフォルトのページサイズではなく huge page を使用してメモリーをゲストに割り当てるよう指示します。

以下のコマンドを実行して、現在の huge page の値を表示します。

```
cat /proc/sys/vm/nr_hugepages
```

手順8.1 huge page の設定

以下の手順例は、huge page を設定するためのコマンドを示しています。

1. 現在の huge page の値を確認します。

```
# cat /proc/meminfo | grep Huge
AnonHugePages:      2048 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

2. Huge page は 2MB 単位で設定されます。huge page の数を 25000 に設定するには、以下のコマンドを使用します。

```
echo 25000 > /proc/sys/vm/nr_hugepages
```

注記

設定を永続化するには、以下の行をゲストマシンの **/etc/sysctl.conf** ファイルに追加します。ここで、X を Huge Page の数値に置き換えます。

```
# echo 'vm.nr_hugepages = X' >> /etc/sysctl.conf
# sysctl -p
```

次に、**transparent_hugepage=never** をゲストの **/etc/grub2.cfg** ファイル内の **/kernel** 行の末尾に追加することで、これをカーネル起動パラメーターに追加します。

3. huge page をマウントします。

```
# mount -t hugetlbfs hugetlbfs /dev/hugepages
```

4. 仮想マシンの XML 設定の **memoryBacking** セクションに次の行を追加します。

```
<hugepages>
  <page size='1' unit='GiB'/>
</hugepages>
```

5. **libvirtd** を再起動してから、以下のコマンドを使って仮想マシンを再起動します。

```
# systemctl start libvirtd
```

```
# virsh start virtual_machine
```

6. **/proc/meminfo** の変更を検証します。

```
# cat /proc/meminfo | grep Huge
AnonHugePages:         0 kB
HugePages_Total:   25000
HugePages_Free:    23425
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

huge page はホストだけでなくゲストにとっても便利ですが、huge page の合計ページ数は必ずホスト内で利用できるページ数より少なくしてください。

8.2.3.3. 起動時またはランタイム時におけるゲストの 1 GB Huge Page の有効化

Red Hat Enterprise Linux 7 システムは、システムの起動時またはランタイム時に割り当てることのできる 2 MB および 1 GB の Huge Page をサポートします。

手順8.2 システム起動時に 1 GB Huge Page の割り当て

1. システムの起動時に、異なるサイズの Huge Page を割り当てるには、以下のコマンドを実行して Huge Page の数を指定します。この例では、4 個の 1 GB の Huge Page と 1024 個の 2 MB の Huge Page を割り当てます。

```
'default_hugepagesz=1G hugepagesz=1G hugepages=4 hugepagesz=2M hugepages=1024'
```

このコマンド行を変更して起動時に割り当てる Huge Page の異なる数を指定できます。

注記

システムの起動時に初めて 1 GB の Huge Page を割り当てる場合は、次の 2 つの手順も実行する必要があります。

2. ホストに 2 MB および 1 GB の Huge Page をマウントします。

```
# mkdir /dev/hugepages1G
# mount -t hugetlbfs -o pagesize=1G none /dev/hugepages1G
# mkdir /dev/hugepages2M
# mount -t hugetlbfs -o pagesize=2M none /dev/hugepages2M
```

3. 仮想マシンの XML 設定の **memoryBacking** セクションに次の行を追加します。

```
<hugepages>
  <page size='1' unit='GiB'/>
</hugepages>
```

4. ゲストで 1GB Huge Page の使用を有効にするために libvirtd を再起動します。

```
# systemctl restart libvirtd
```

手順8.3 ランタイム時の 1 GB Huge Page の割り当て

1 GB の Huge Page は、ランタイム時に割り当てることもできます。ランタイム時の割り当てにより、システム管理者はそれらのページの割り当て元となる NUMA ノードを選択できます。ただし、ランタイム時のページ割り当ては、メモリーの断片化により、システム起動時の割り当てに比べて、割り当てが失敗する確率が大きくなります。

1. ランタイム時に Huge Page の異なるサイズを割り当てるには、以下のコマンドを使用します。その際、Huge Page の数や、それらのページの割り当て元となる NUMA ノード、および Huge Page サイズの値を置き換えます。

```
# echo 4 > /sys/devices/system/node/node1/hugepages/hugepages-
1048576kB/nr_hugepages
# echo 1024 > /sys/devices/system/node/node3/hugepages/hugepages-
2048kB/nr_hugepages
```

このサンプルコマンドにより、**node1** から 4 個の 1 GB の Huge Page が割り当てられ、 **node3** から 1024 個の 2 MB の Huge Page が割り当てられます。

上記のコマンドを使用すると、これらの huge page 設定をホストシステムの空きメモリーの量に応じていつでも変更できます。

注記

ランタイム時に初めて 1 GB の Huge Page を割り当てる場合は、次の 2 つの手順も実行する必要があります。

2. ホストに 2 MB および 1 GB の Huge Page をマウントします。

```
# mkdir /dev/hugepages1G
# mount -t hugetlbfs -o pagesize=1G none /dev/hugepages1G
# mkdir /dev/hugepages2M
# mount -t hugetlbfs -o pagesize=2M none /dev/hugepages2M
```

3. 仮想マシンの XML 設定の **memoryBacking** セクションに次の行を追加します。

```
<hugepages>
  <page size='1' unit='GiB'/>
</hugepages>
```

4. ゲストで 1GB Huge Page の使用を有効にするために libvirtd を再起動します。

```
# systemctl restart libvirtd
```

8.3. KSM (KERNEL SAME-PAGE MERGING)

KVM ハイパーバイザーが使用する Kernel same-page Merging (KSM) により、KVM ゲストは同一メモリーページを共有することが可能になります。このような共有ページは通常、共通のライブラリーか、

使用頻度の高い同一データです。KSM はメモリー重複を防ぐことで同一または類似のゲストオペレーティングシステムのゲストの密度を高めることができます。

最近のオペレーティングシステムでは共有メモリーという概念が一般的になってきました。たとえば、プログラムが初めて起動する際に、そのプログラムはその親プログラムとすべてのメモリーを共有します。子プログラムまたは親プログラムのいずれかがメモリーの変更を試行すると、カーネルによって新しいメモリー領域が割り当てられ、元のコンテンツがコピーされます。これにより、プログラムがこの新たな範囲を変更できるようになります。これはコピーオンライト (copy on write) と呼ばれます。

KSM はこの概念を逆方向で用いる Linux の機能になります。カーネルにすでに実行中の複数のプログラムを検査させ、メモリーの比較を行います。そのメモリーの範囲またはページが同一の場合は、その複数存在している同じメモリーページを 1 つのページに減らします。このページにコピーオンライトのマークが付けられます。ページの内容がゲスト仮想マシンによって変更される場合は、そのゲスト用に新しいページが作成されます。

KSM は KVM を使った仮想化に便利です。ゲスト仮想マシンの起動時点では、ホストの **qemu-kvm** プロセスからのメモリーしか継承しません。同じオペレーティングシステムやアプリケーションを複数のゲストが実行している場合、ゲストの実行時にゲストのオペレーティングシステムイメージのコンテンツが共有されるようになります。KSM の機能によって KVM は同一ゲストのメモリー領域が共有されるよう要求することができるようになります。

KSM によってメモリーの速度やその用途が広がります。KSM を使用すると、共通の処理データはキャッシュやメインメモリーに格納されます。これにより KVM ゲストのキャッシュミスが低減されるため、一部のアプリケーションやオペレーティングシステムのパフォーマンスを向上させることができます。また、メモリーを共有することによりゲストの全体的なメモリー使用を抑えるため、より多くのリソースを有効に活用できるようになります。

注記

Red Hat Enterprise Linux 7 より、KSM は NUMA 対応になりました。これにより、ページコアレッシングの実行時に NUMA ローカリティーが考慮されることになり、リモートノードに移行されるページに関連するパフォーマンスの低下を防ぐことができるようになります。Red Hat は、KSM の使用時にはノード間のメモリーマージを控えることを推奨します。KSM が使用中の場合には、**/sys/kernel/mm/ksm/merge_across_nodes** パラメーターを ***0*** に変更し、複数の NUMA ノード間でのページのマージを防ぎます。これは、**virsh node-memory-tune --shm-merge-across-nodes 0** コマンドを使用して実行できます。多量のノード間マージが実行すると、カーネルメモリーのアカウンティング統計は相反する結果となる可能性があります。そのため、numad も KSM デーモンが多量のメモリーをマージした後に混乱する可能性があります。システムに大量の空きメモリーがある場合、KSM デーモンをオフにし、無効にすることでパフォーマンスを向上させることができます。NUMA の詳細は9章*NUMA*を参照してください。

重要

KSM を考えに含めなくてもコミットする RAM に対して swap サイズが十分であることを確認してください。KSM は同一または類似ゲストの RAM 使用量を低減します。十分なスワップ領域がなくても KSM を使ってゲストをオーバーコミットすることはおそらく可能ですが、ゲスト仮想マシンのメモリー使用によってページが共有されなくなる可能性があるため推奨されません。

Red Hat Enterprise Linux では、KSM の管理に以下の 2 種類の方法を使用しています。

- **ksm サービス** は、KSM カーネルスレッドの起動と停止を行います。
- **ksmtuned サービス** は、**ksm** サービスの制御と調整を行い、same-page merging を動的に管

理します。**ksmtuned** は **ksm** サービスを起動し、メモリー共有が必要ない場合には **ksm** サービスを停止します。新規のゲストが作成された場合またはゲストが破棄された場合には、***retune*** パラメーターで **ksmtuned** に実行の指示を出さなければなりません。

いずれのサービスも標準のサービス管理ツールで制御されます。

注記

Red Hat Enterprise Linux 6.7 では、KSM はデフォルトで無効になっています。

8.3.1. KSM サービス

- **ksm** サービスは qemu-kvm パッケージに含まれています。
- **ksm** サービスを起動していない場合、Kernel same-page merging (KSM) により 2000 ページしか共有されません。このデフォルト値では、メモリー節約で得られる利点が限られます。
- **ksm** サービスを起動すると、KSM により最大でホストシステムのメインメモリーの 50% まで共有されるようになります。**ksm** サービスを起動して KSM がより多くのメモリーを共有できるようにしてください。

```
# systemctl start ksm
Starting ksm:                                [  OK  ]
```

ksm サービスをデフォルトのスタートアップ順序に追加することができます。systemctl コマンドを使って **ksm** サービスを永続化します。

```
# systemctl enable ksm
```

8.3.2. KSM チューニングサービス

ksmtuned サービスは、**ksm** をループおよび調整して kernel same-page merging (KSM) の設定を微調整します。また、ゲスト仮想マシンが作成または破棄された場合は、そのことが libvirt によって **ksmtuned** サービスに通知されます。**ksmtuned** サービスにオプションはありません。

```
# systemctl start ksmtuned
Starting ksmtuned:                           [  OK  ]
```

ksmtuned サービスは ***retune*** パラメーターを使って調整することができます。パラメーターの指示によって **ksmtuned** は手動によるチューニング機能を実行します。

/etc/ksmtuned.conf ファイルは **ksmtuned** サービスの設定ファイルになります。デフォルトの **ksmtuned.conf** ファイルの出力を以下に示します。

```
# Configuration file for ksmtuned.
# How long ksmtuned should sleep between tuning adjustments
# KSM_MONITOR_INTERVAL=60

# Millisecond sleep between ksm scans for 16Gb server.
# Smaller servers sleep more, bigger sleep less.
# KSM_SLEEP_MSEC=10

# KSM_NPAGES_BOOST - is added to the `npages` value, when `free memory` is less than `thres`.
```

```
# KSM_NPAGES_BOOST=300

# KSM_NPAGES_DECAY - is the value given is subtracted to the `npages` value, when `free
memory` is greater than `thres`.
# KSM_NPAGES_DECAY=-50

# KSM_NPAGES_MIN - is the lower limit for the `npages` value.
# KSM_NPAGES_MIN=64

# KSM_NPAGES_MAX - is the upper limit for the `npages` value.
# KSM_NPAGES_MAX=1250

# KSM_THRES_COEF - is the RAM percentage to be calculated in parameter `thres`.
# KSM_THRES_COEF=20

# KSM_THRES_CONST - If this is a low memory system, and the `thres` value is less than
`KSM_THRES_CONST`, then reset `thres` value to `KSM_THRES_CONST` value.
# KSM_THRES_CONST=2048

# uncomment the following to enable ksmtuned debug information
# LOGFILE=/var/log/ksmtuned
# DEBUG=1
```

/etc/ksmtuned.conf ファイルの **npages** により、**ksmd** デーモンが非アクティブになるまでに **ksm** がスキャンするページ数が設定されます。この値は、**/sys/kernel/mm/ksm/pages_to_scan** ファイルでも設定されます。

KSM_THRES_CONST の値は、**ksm** をアクティブ化する際のしきい値として使われる、利用可能なメモリー容量を表します。以下のいずれかの状況になった場合に、**ksmd** がアクティブ化されます。

- 空きメモリー容量が **KSM_THRES_CONST** で設定されたしきい値を下回った場合
- コミットしたメモリー容量と **KSM_THRES_CONST** のしきい値の合計が、総メモリー容量を超えた場合

8.3.3. KSM の変数とモニタリング

Kernel same-page merging (KSM) はモニタリングデータを **/sys/kernel/mm/ksm/** ディレクトリーに格納します。このディレクトリー内のファイルはカーネルによって更新されるため、KSM の使用状況と統計値の正確な記録となります。

上述のように、以下に示す変数は **/etc/ksmtuned.conf** ファイルでも設定可能です。

/sys/kernel/mm/ksm/ に含まれるファイル:

full_scans
実行された完全スキャン数

merge_across_nodes
異なる NUMA ノードからページをマージできるかどうかを指定します。

pages_shared
共有されたページの合計数

pages_sharing
現在共有されているページ数

pages_to_scan
スキャンされなかったページ数

pages_unshared
共有されなくなったページ数

pages_volatile
揮発性のページ数

run
KSM プロセスが実行しているかどうか

sleep_millisecs
スリープのミリ秒数

これらの変数は **virsh node-memory-tune** コマンドを使用して手動で調整できます。たとえば、共有メモリーサービスがスリープ状態になる前にスキャンするページの数を指定します。

```
# virsh node-memory-tune --shm-pages-to-scan number
```

/etc/ksmtuned.conf ファイルに ***DEBUG=1*** の行を追加すると、KSM チューニングのアクティビティーが **/var/log/ksmtuned** ログファイルに格納されます。***LOGFILE*** パラメーターを使用するとログファイルの場所を変更することができます。ただし、ログファイルの場所の変更は、SELinux 設定に特殊な構成を必要とする場合があるためお勧めできません。

8.3.4. KSM の非アクティブ化

Kernel same-page merging (KSM) にはパフォーマンス上のオーバーヘッドがあり、特定の環境やホストシステムには負荷が大きすぎる場合があります。また、KSM はサイドチャネルを生成する恐れがあり、ゲスト間に情報を漏洩するために使用される可能性があります。これが懸念される場合には、ゲストごとに KSM を無効にすることができます。

ksmtuned と **ksm** サービスを停止して、KSM を非アクティブ化することができます。ただし、再起動後は元に戻ります。KSM を非アクティブ化するには、ターミナルから root として以下のコマンドを実行します。

```
# systemctl stop ksmtuned
Stopping ksmtuned:                         [  OK  ]
# systemctl stop ksm
Stopping ksm:                              [  OK  ]
```

ksmtuned と **ksm** を停止すると KSM は非アクティブ化されますが、再起動後は元に戻ります。KSM を完全に非アクティブ化するには、**systemctl** コマンドを使用します。

```
# systemctl disable ksm
# systemctl disable ksmtuned
```

KSM を無効にする場合、KSM を非アクティブ化する前に共有されていたすべてのメモリーページは、引き続き共有されます。システムからすべての PageKSM を削除するには、以下のコマンドを使用します。

```
# echo 2 >/sys/kernel/mm/ksm/run
```

これを実施した後に、**khugepaged** デーモンは KVM ゲストの物理メモリーに Transparent Hugepage を再ビルドできます。# **echo 0 >/sys/kernel/mm/ksm/run** を使用すると、KSM は停止しますが、以前に作成されたすべての KSM ページは共有されたままです (これは # **systemctl stop ksmtuned** コマンドと同じです)。

第9章 NUMA

従来の AMD64 および Intel 64 システムでは、メモリーはすべて均等に全 CPU からアクセスできるようになっていました。どの CPU がその操作を実行するかに関わらず、UMA (Uniform Memory Access) と呼ばれるアクセスタイムは同じになります。

最新の AMD64 および Intel 64 プロセッサーでは、この動作が異なってきています。NUMA (Non-Uniform Memory Access) では、システムのメモリーが複数の NUMA **ノード** に分類されます。これらのノードは、ソケット、またはシステムメモリーのローカルのサブセットに対するアクセスレイテンシーが同じである特定セットの CPU に対応しています。

本章では、仮想化環境におけるメモリーの割り当ておよび NUMA チューニング設定について説明します。

9.1. NUMA メモリー割り当てのポリシー

システム内のノードからどのようにメモリーを割り当てるかは、以下のポリシーで定義されます。

Strict

Strict ポリシーでは、目的のノードでメモリーを割り当てられない場合は割り当てに失敗することになります。

メモリーモード属性を定義せずに NUMA ノードセットのリストを指定すると、デフォルトで ***strict*** モードが選択されます。

Interleave

メモリーページはノードセットで指定されたノード全体に割り当てられますが、割り当てはラウンドロビン方式で行われます。

Preferred

単一の優先メモリーノードからのみメモリーの割り当てが行われます。十分なメモリーが使用できない場合には、他のノードからメモリーを割り当てることができます。

目的のポリシーを有効にするには、ドメイン XML ファイルの **<memory mode>** 要素の値にそのポリシーを設定します。

```
<numatune>
      <memory mode='preferred' nodeset='0'>
</numatune>
```

重要

strict モードにおいてメモリーがオーバーコミットされ、ゲストに十分な Swap 領域がない場合、カーネルはゲストのプロセスの一部を強制終了して追加のメモリーを取得します。この状況を回避するために、Red Hat では ***preferred*** ポリシーによる割り当てを使用し、単一のノードセットを指定する (例: nodeset=「0」) ことを推奨します。

9.2. NUMA の自動負荷分散

NUMA の自動負荷分散により、NUMA ハードウェアシステムで実行するアプリケーションのパフォーマンスが向上します。この機能は Red Hat Enterprise Linux 7 システムではデフォルトで有効にされます。

通常アプリケーションは、スレッドがスケジュールされるのと同じ NUMA ノード上でプロセスのスレッドがメモリーにアクセスする場合に最適に実行されます。NUMA の自動負荷分散は、スレッドがアクセスしているメモリーの近くにタスク (スレッドまたはプロセス) を移行します。さらに、アプリケーションデータを、これを参照するタスクの近くにあるメモリーに移動します。これはすべて、NUMA 自動負荷分散がアクティブな場合にカーネルによって自動的に実行されます。

NUMA の自動負荷分散は、数多くのアルゴリズムおよびデータ構造を使用します。これらは NUMA の自動負荷分散がシステム上で有効な場合にのみアクティブになり、割り当てられます。

- Periodic NUMA unmapping of process memory (プロセスメモリーの定期的な NUMA マッピング解除)
- NUMA hinting fault (NUMA ヒンティングフォールト)
- Migrate-on-Fault (MoF) - メモリーを使用するプログラムが実行される場所にメモリーを移動します。
- task_numa_placement - 実行中のプログラムをそれらのメモリーの近くに移動します。

9.2.1. NUMA の自動負荷分散の設定

Red Hat Enterprise Linux 7 では、NUMA の自動負荷分散はデフォルトで有効にされており、この機能は NUMA プロパティーが設定されたハードウェア上で起動する際に自動的にアクティブにされます。

NUMA の自動負荷分散は、以下の条件の両方を満たす場合に有効にされます。

- **# numactl --hardware** は複数のノードを表示します。
- **# cat /proc/sys/kernel/numa_balancing** は **1** を示します。

NUMA の手動によるアプリケーションのチューニングは NUMA の自動負荷分散をオーバーライドし、メモリーの定期的なマッピング解除、NUMA フォールト、移行、およびそれらのアプリケーションの自動の NUMA 配置を無効にします。

システム全体での手動の NUMA チューニングがより望ましい場合があります。

NUMA の自動負荷分散を無効にするには、以下のコマンドを使用します。

```
# echo 0 > /proc/sys/kernel/numa_balancing
```

NUMA の自動負荷分散を有効にするには、以下のコマンドを使用します。

```
# echo 1 > /proc/sys/kernel/numa_balancing
```

9.3. LIBVIRT の NUMA チューニング

通常 NUMA システム上の最高のパフォーマンスは、単一 NUMA ノードのリソース量にゲストのサイズを制限することによって達成できます。複数の NUMA ノード間でリソースを不必要に分割することを避けてください。

numastat ツールを使用して、プロセスおよびオペレーティングシステムの NUMA ノードごとのメモリー統計を表示します。

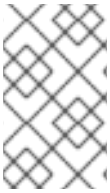
以下の例では、**numastat** ツールは、複数の NUMA ノード間のメモリー配置が最適ではない 4 つの仮想マシンを示しています。

```
# numastat -c qemu-kvm

Per-node process memory usage (in MBs)
PID              Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
51722 (qemu-kvm)     68     16    357   6936      2      3    147    598  8128
51747 (qemu-kvm)    245     11      5     18   5172   2532      1     92  8076
53736 (qemu-kvm)     62    432   1661    506   4851    136     22    445  8116
53773 (qemu-kvm)   1393      3      1      2     12      0      0   6702  8114
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total              1769    463   2024   7462  10037   2672    169   7837 32434
```

numad を実行して、ゲストの CPU とメモリーリソースを自動的に調整します。

次に **numastat -c qemu-kvm** を再び実行して実行中の **numad** の結果を表示します。以下の出力は、リソースが調整されていることを示しています。

```
# numastat -c qemu-kvm

Per-node process memory usage (in MBs)
PID              Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
51747 (qemu-kvm)      0      0      7      0   8072      0      1      0  8080
53736 (qemu-kvm)      0      0      7      0      0      0   8113      0  8120
53773 (qemu-kvm)      0      0      7      0      0      0      1   8110  8118
59065 (qemu-kvm)      0      0   8050      0      0      0      0      0  8051
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total                 0      0   8072      0   8072      0   8114   8110 32368
```



注記

-c を指定して **numastat** を実行することにより、簡易出力が表示されます。**-m** オプションを追加すると、システム全体でのメモリー情報のノード別の内訳が出力に追加されます。詳細は、**numastat** の man ページを参照してください。

9.3.1. ホスト NUMA ノードごとのメモリー監視

nodestats.py スクリプトを使用して、ホスト上の各 NUMA ノードの合計メモリーと空きメモリーを報告できます。また、このスクリプトは実行中の各ドメインの特定のホストノードに対して厳密にバインドされているメモリーの量をレポートします。以下は例になります。

```
# /usr/share/doc/libvirt-python-2.0.0/examples/nodestats.py
NUMA stats
NUMA nodes:     0       1       2       3
MemTotal:       3950    3967    3937    3943
MemFree:        66      56      42      41
Domain 'rhel7-0':
        Overall memory: 1536 MiB
```

```
Domain 'rhel7-1':
        Overall memory: 2048 MiB
Domain 'rhel6':
        Overall memory: 1024 MiB nodes 0-1
        Node 0: 1024 MiB nodes 0-1
Domain 'rhel7-2':
        Overall memory: 4096 MiB nodes 0-3
        Node 0: 1024 MiB nodes 0
        Node 1: 1024 MiB nodes 1
        Node 2: 1024 MiB nodes 2
        Node 3: 1024 MiB nodes 3
```

この例は、それぞれに約 4GB の合計 RAM (***MemTotal***) が含まれる 4 つのホスト NUMA ノードを示しています。ほぼすべてのメモリーが各ドメインで消費されています (***MemFree***)。4 つのドメイン (仮想マシン) が実行されています。ドメイン *'rhel7-0'* には、特定のホスト NUMA ノードにピニングされていない 1.5GB メモリーがあります。ドメイン *'rhel7-2'* には 4GB メモリーがあり、4 つの NUMA ノードはホストノードに 1:1 でピニングされています。

ホスト NUMA ノードの統計値を出力するには、ご使用の環境用の **nodestats.py** スクリプトを作成します。サンプルスクリプトは、**/usr/share/doc/libvirt-python-*version*/examples/nodestats.py** の libvirt-python パッケージファイルにあります。**rpm -ql libvirt-python** コマンドを使用して、スクリプトへの実際のパスを表示することができます。

9.3.2. NUMA の vCPU ピニング

vCPU ピニングは、ベアメタルシステム上でのタスクを固定する場合と同様の利点を提供します。vCPU はホストオペレーティングシステム上でユーザー領域タスクとして実行されるため、ピニングによりキャッシュ効率が増大します。この 1 つの例となるのは、すべての vCPU スレッドが同じ物理ソケット上で実行されるため、それらが L3 キャッシュドメインを共有する環境です。

注記

Red Hat Enterprise Linux バージョン 7.0 から 7.2 では、アクティブな vCPU のみをピニングできます。ただし、Red Hat Enterprise Linux 7.3 では、非アクティブな vCPU のピニングも実行できます。

vCPU ピニングを **numatune** と組み合わせることにより、NUMA ミスが回避されます。NUMA ミスによるパフォーマンスへの影響は大きく、通常はパフォーマンスが 10% 以上低下します。vCPU ピニングと **numatune** は一緒に設定する必要があります。

仮想マシンがストレージまたはネットワーク I/O タスクを実行している場合、I/O アダプターに物理的に接続されている同じ物理ソケットにすべての vCPU とメモリーを固定すると効果的です。

注記

lstopo ツールを使用して NUMA トポロジーを仮想化できます。さらに、このツールは vCPU が同じ物理ソケットのコアにバインドされていることを検証する際にも役立ちます。**lstopo** の詳細は、ナレッジベース https://access.redhat.com/site/solutions/62879 を参照してください。

重要

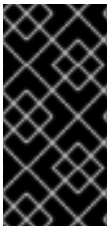
ピニングは、物理コアより多くの vCPU がある場合には複雑化の原因になります。

以下の XML 設定例では、ドメインプロセスが物理 CPU 0 から 7 に固定されています。vCPU スレッドはそれぞれ独自の cpuset に固定されています。たとえば、vCPU0 は物理 CPU0 に、vCPU1 は物理 CPU 1 に固定されています。

```
<vcpu cpuset='0-7'>8</vcpu>
    <cputune>
        <vcpupin vcpu='0' cpuset='0'/>
        <vcpupin vcpu='1' cpuset='1'/>
        <vcpupin vcpu='2' cpuset='2'/>
        <vcpupin vcpu='3' cpuset='3'/>
        <vcpupin vcpu='4' cpuset='4'/>
        <vcpupin vcpu='5' cpuset='5'/>
        <vcpupin vcpu='6' cpuset='6'/>
        <vcpupin vcpu='7' cpuset='7'/>
    </cputune>
```

vcpu と vcpupin タグ間には直接的な関係があります。vcpupin オプションを指定しないと、値は自動的に確定され、親となる vcpu タグオプションから継承されます。以下の設定では **vcpu 5** の**<vcpupin>** がないことを示しています。したがって、**vCPU5** は親タグの **<vcpu>** で指定しているように物理 CPU の 0 から 7 に固定される可能性があります。

```
<vcpu cpuset='0-7'>8</vcpu>
    <cputune>
        <vcpupin vcpu='0' cpuset='0'/>
        <vcpupin vcpu='1' cpuset='1'/>
        <vcpupin vcpu='2' cpuset='2'/>
        <vcpupin vcpu='3' cpuset='3'/>
        <vcpupin vcpu='4' cpuset='4'/>
        <vcpupin vcpu='6' cpuset='6'/>
        <vcpupin vcpu='7' cpuset='7'/>
    </cputune>
```



重要

最適なパフォーマンスを確実に実現するには、**<vcpupin>**、**<numatune>**、および **<emulatorpin>** を共に設定する必要があります。**<numatune>** タグの詳細は、「ドメインプロセス」を参照してください。**<emulatorpin>** タグの詳細は、「emulatorpin の使用」を参照してください。

9.3.3. ドメインプロセス

Red Hat Enterprise Linux で規定されている通り、libvirt ではドメインプロセスのメモリーバインディングのポリシー設定に libnuma を使用します。これらのポリシーのノードセットは、**static** (ドメイン XML に指定) か、または **auto** (numad のクエリーで設定) のいずれかに設定できます。**<numatune>** タグ内にこれらを設定する方法は、以下の XML 設定例を参照してください。

```
<numatune>
    <memory mode='strict' placement='auto'/>
</numatune>
```

```
<numatune>
    <memory mode='strict' nodeset='0,2-3'/>
</numatune>
```

libvirt は、**sched_setaffinity(2)** を使用してドメインプロセスの CPU バインディングポリシーを設定します。cpuset オプションは。**static** (ドメイン XML で指定) または **auto** (numad のクエリーで設定) にできます。**<vcpu>** タグ内に設定する方法は、以下の XML 設定例を参照してください。

```
<vcpu placement='auto'>8</vcpu>
```

```
<vcpu placement='static' cpuset='0-10,^5'>8</vcpu>
```

<vcpu> および **<numatune>** に使用する配置モード間には暗黙的な継承ルールがあります。

- **<numatune>** の配置モードは **<vcpu>** と同じ配置モードにデフォルト設定されます。または、**<nodeset>** を指定した場合は*static* に設定されます。
- 同様に、**<vcpu>** の配置モードでは **<numatune>** と同じ配置モードにデフォルト設定されます。または、**<cpuset>** を指定した場合は *static* に設定されます。

つまり、ドメインプロセスのメモリーチューニングと CPU チューニングは別々に指定し、定義することができますが、同時に相互の配置モードに依存するように設定することもできます。

さらに、起動時にすべての vCPU を固定せずに選択した数の vCPU を起動できるように numad を使用してシステムを設定することもできます。

たとえば、32 の vCPU が設定されたシステムで 8 つの vCPU のみを有効にするには、以下のように XML を設定します。

```
<vcpu placement='auto' current='8'>32</vcpu>
```

注記

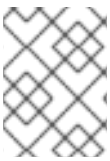
vcpu および numatune に関する詳細は、http://libvirt.org/formatdomain.html#elementsCPUAllocation および http://libvirt.org/formatdomain.html#elementsNUMATuning を参照してください。

9.3.4. ドメインの vCPU スレッド

ドメインプロセスのチューニングのほかにも、libvirt では XML 設定内の vcpu の各スレッドにピニングポリシーの設定を許可します。**<cputune>** タグ内に各 vcpu スレッドのピニングポリシーを設定します。

```
<cputune>
      <vcpupin vcpu="0" cpuset="1-4,^2"/>
      <vcpupin vcpu="1" cpuset="0,1"/>
      <vcpupin vcpu="2" cpuset="2,3"/>
      <vcpupin vcpu="3" cpuset="0,4"/>
</cputune>
```

このタグでは、libvirt は cgroup または **sched_setaffinity(2)** のいずれかを使って vcpu スレッドを指定の cpuset に固定しています。



注記

<cputune> の詳細は、http://libvirt.org/formatdomain.html#elementsCPUTuning を参照してください。

さらに、単一の NUMA ノードよりも多くの vCPU を持つ仮想マシンをセットアップする必要がある場合、ゲストがホスト上で NUMA トポロジーを検知できるようにホストを設定します。これにより、CPU、メモリーおよび NUMA ノードの 1:1 のマッピングが可能になります。たとえば、これは 4 つの vCPU と 6 GB メモリーを持つゲストと、以下の NUMA 設定を持つホストに適用できます。

```
4 available nodes (0-3)
Node 0: CPUs 0 4, size 4000 MiB
Node 1: CPUs 1 5, size 3999 MiB
Node 2: CPUs 2 6, size 4001 MiB
Node 3: CPUs 0 4, size 4005 MiB
```

このシナリオでは、以下のドメイン XML 設定を使用します。

```
<cputune>
 <vcpupin vcpu="0" cpuset="1"/>
 <vcpupin vcpu="1" cpuset="5"/>
 <vcpupin vcpu="2" cpuset="2"/>
 <vcpupin vcpu="3" cpuset="6"/>
</cputune>
<numatune>
  <memory mode="strict" nodeset="1-2"/>
</numatune>
<cpu>
 <numa>
  <cell id="0" cpus="0-1" memory="3" unit="GiB"/>
  <cell id="1" cpus="2-3" memory="3" unit="GiB"/>
 </numa>
</cpu>
```

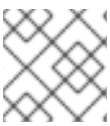
9.3.5. キャッシュ割り当て技術を使用したパフォーマンスの向上

特定の CPU モデルで、カーネルが提供する CAT (Cache Allocation Technology) を利用できます。これにより、リアルタイムのパフォーマンスを向上させる vCPU スレッドのホスト CPU のキャッシュの一部を割り当てることができるようになります。

cachetune のタグ内の vCPU キャッシュの割り当てを設定する例は、以下の XML 設定例を参照してください。

```
<domain>
  <cputune>
    <cachetune vcpus='0-1'>
      <cache id='0' level='3' type='code' size='3' unit='MiB'/>
      <cache id='0' level='3' type='data' size='3' unit='MiB'/>
    </cachetune>
  </cputune>
</domain>
```

前述の XML ファイルでは、vCPUs 0 および 1 用のスレッドに、最初の L3 キャッシュ (level='3' id='0') から 3 MiB が割り当てられるように設定されています (L3CODE 用および L3DATA 用)。



注記

1 つの仮想マシンに、複数の **<cachetune>** 要素を持つことができます。

詳細は、アップストリームの **libvirt** ドキュメント の **cachetune** を参照してください。

9.3.6. emulatorpin の使用

ドメインプロセスのピニングポリシーを調整する別の方法として、**<cputune>** 内で **<emulatorpin>** タグを使用する方法があります。

<emulatorpin> タグは、**エミュレーター** (vCPU を含まないドメインのサブセット) が固定されるホスト物理 CPU を指定します。**<emulatorpin>** タグは、エミュレーターのスレッドプロセスに正確な親和性 (アフィニティー) を設定する方法を提供します。結果として vhost スレッドは物理 CPU およびメモリーの同じサブセットで実行されるため、キャッシュの局所性 (ローカリティー) によるメリットがあります。たとえば、以下のようになります。

```
<cputune>
      <emulatorpin cpuset="1-3"/>
</cputune>
```

注記

Red Hat Enterprise Linux 7 では、NUMA の自動負荷分散はデフォルトで有効にされています。NUMA の自動負荷分散は、vhost-net エミュレータースレッドと vCPU タスクとの連携がより正確に行われるため、**<emulatorpin>** を手動で調整する必要性を軽減します。NUMA の自動負荷分散についての詳細は、「NUMA の自動負荷分散」 を参照してください。

9.3.7. virsh による vCPU ピニングのチューニング

重要

以下に示すのは説明を目的としたコマンド例です。実際には、使用する環境に適した値を入力する必要があります。

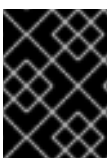
以下の **virsh** コマンドの例では、ID が 1 となる vcpu スレッド *rhel7* を物理 CPU 2 に固定します。

```
% virsh vcpupin rhel7 1 2
```

さらに、**virsh** コマンドで 現在の vcpu ピニング設定を取得することもできます。以下は例になります。

```
% virsh vcpupin rhel7
```

9.3.8. virsh を使用したドメインプロセスの CPU ピニングのチューニング



重要

以下に示すのは説明を目的としたコマンド例です。実際には、使用する環境に適した値を入力する必要があります。

emulatorpin オプションでは、各ドメインプロセスに関連付けられたスレッドに CPU アフィニティーの設定を適用します。詳細のピニングを実行するには、**virsh vcpupin** (前述) と **virsh emulatorpin** の両方を各ゲストに使用する必要があります。例を示します。

```
% virsh emulatorpin rhel7 3-4
```

9.3.9. virsh を使用したドメインプロセスのメモリーポリシーのチューニング

ドメインプロセスのメモリーには動的なチューニングを実行できます。以下のコマンド例を参照してください。

```
% virsh numatune rhel7 --nodeset 0-10
```

これらのコマンドの詳細な使用例については、**virsh** の man ページを参照してください。

9.3.10. ゲスト NUMA トポロジー

ゲスト NUMA トポロジーは、ゲスト仮想マシンの XML の **<cpu>** タグ内にある **<numa>** タグを使用して指定できます。以下の例を参照し、値を適宜置き換えてください。

```
<cpu>
      ...
  <numa>
    <cell cpus='0-3' memory='512000'/>
    <cell cpus='4-7' memory='512000'/>
  </numa>
  ...
</cpu>
```

それぞれの **<cell>** 要素は NUMA セルまたは NUMA ノードを指定します。 **cpus** は、ノードの一部である CPU または CPU の範囲を指定します。**memory** はノードメモリーをキビバイト単位で指定します (1024 バイトのブロック)。それぞれのセルまたはノードには、0 から始まる昇順で **cellid** または **nodeid** が割り当てられます。

重要

ゲスト仮想マシンの NUMA トポロジーを、CPU ソケット、コアおよびスレッドの設定されたトポロジーで変更する場合、単一ソケットに属するコアとスレッドが同じ NUMA ノードに割り当てられていることを確認します。同じソケットのスレッドまたはコアが異なる NUMA ノードに割り当てられている場合は、ゲストは起動に失敗する可能性があります。

警告

Red Hat Enterprise Linux 7 では、Huge Page と同時にゲストの NUMA トポロジーを使用することはサポートされておらず、Red Hat Virtualization、Red Hat OpenStack Platform などのレイヤード製品でのみ使用できます。

9.3.11. PCI デバイスの NUMA ノードの局所性 (ローカリティー)

新規の仮想マシンを起動する際、ホスト NUMA トポロジーと PCI デバイスの NUMA ノードとの関連生の両方について理解しておくことは大切です。PCI パススルーが要求される場合、ゲストがメモリーパフォーマンスの最適化に向けて正しい NUMA ノードに固定されるようにするためです。

たとえば、ゲストが NUMA ノード 0-1 に固定するが、その PCI デバイスのいずれかがノード 2 に関連付けられる場合、ノード間のデータ送信にはいくらかの時間がかかります。

Red Hat Enterprise Linux 7.1 以降では、libvirt はゲスト XML で PCI デバイスについての NUMA ノードの局所性 (ローカリティー) をレポートし、管理アプリケーションでの適切なパフォーマンス関連の決定ができるようにします。

この情報は、**/sys/devices/pci*/*/numa_node** の **sysfs** ファイルに表示されます。これらの設定を検証する 1 つの方法は、**lstopo** ツールを使用して **sysfs** データをレポートする方法です。

```
# lstopo-no-graphics
Machine (126GB)
  NUMANode L#0 (P#0 63GB)
    Socket L#0 + L3 L#0 (20MB)
      L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0 + PU L#0 (P#0)
      L2 L#1 (256KB) + L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1 + PU L#1 (P#2)
      L2 L#2 (256KB) + L1d L#2 (32KB) + L1i L#2 (32KB) + Core L#2 + PU L#2 (P#4)
      L2 L#3 (256KB) + L1d L#3 (32KB) + L1i L#3 (32KB) + Core L#3 + PU L#3 (P#6)
      L2 L#4 (256KB) + L1d L#4 (32KB) + L1i L#4 (32KB) + Core L#4 + PU L#4 (P#8)
      L2 L#5 (256KB) + L1d L#5 (32KB) + L1i L#5 (32KB) + Core L#5 + PU L#5 (P#10)
      L2 L#6 (256KB) + L1d L#6 (32KB) + L1i L#6 (32KB) + Core L#6 + PU L#6 (P#12)
      L2 L#7 (256KB) + L1d L#7 (32KB) + L1i L#7 (32KB) + Core L#7 + PU L#7 (P#14)
    HostBridge L#0
      PCIBridge
        PCI 8086:1521
          Net L#0 "em1"
        PCI 8086:1521
          Net L#1 "em2"
        PCI 8086:1521
          Net L#2 "em3"
        PCI 8086:1521
          Net L#3 "em4"
      PCIBridge
        PCI 1000:005b
          Block L#4 "sda"
          Block L#5 "sdb"
          Block L#6 "sdc"
          Block L#7 "sdd"
      PCIBridge
        PCI 8086:154d
          Net L#8 "p3p1"
        PCI 8086:154d
          Net L#9 "p3p2"
      PCIBridge
        PCIBridge
          PCIBridge
            PCIBridge
              PCI 102b:0534
                GPU L#10 "card0"
                GPU L#11 "controlD64"
```

```
    PCI 8086:1d02
  NUMANode L#1 (P#1 63GB)
   Socket L#1 + L3 L#1 (20MB)
    L2 L#8 (256KB) + L1d L#8 (32KB) + L1i L#8 (32KB) + Core L#8 + PU L#8 (P#1)
    L2 L#9 (256KB) + L1d L#9 (32KB) + L1i L#9 (32KB) + Core L#9 + PU L#9 (P#3)
    L2 L#10 (256KB) + L1d L#10 (32KB) + L1i L#10 (32KB) + Core L#10 + PU L#10 (P#5)
    L2 L#11 (256KB) + L1d L#11 (32KB) + L1i L#11 (32KB) + Core L#11 + PU L#11 (P#7)
    L2 L#12 (256KB) + L1d L#12 (32KB) + L1i L#12 (32KB) + Core L#12 + PU L#12 (P#9)
    L2 L#13 (256KB) + L1d L#13 (32KB) + L1i L#13 (32KB) + Core L#13 + PU L#13 (P#11)
    L2 L#14 (256KB) + L1d L#14 (32KB) + L1i L#14 (32KB) + Core L#14 + PU L#14 (P#13)
    L2 L#15 (256KB) + L1d L#15 (32KB) + L1i L#15 (32KB) + Core L#15 + PU L#15 (P#15)
   HostBridge L#8
    PCIBridge
     PCI 1924:0903
      Net L#12 "p1p1"
     PCI 1924:0903
      Net L#13 "p1p2"
    PCIBridge
     PCI 15b3:1003
      Net L#14 "ib0"
      Net L#15 "ib1"
      OpenFabrics L#16 "mlx4_0"
```

この出力は以下の点を示しています。

- NIC **em*** およびディスク **sd*** は NUMA ノード 0 およびコア 0、2、4、6、8、10、12、14 に接続されます。
- NIC **p1*** および **ib*** は NUMA ノード 1 およびコア 1、3、5、7、9、11、13、15 に接続されます。

9.4. NUMA 対応 KSM (KERNEL SAMEPAGE MERGING)

Kernel SamePage Merging (KSM) により、仮想マシンは同一のメモリーページを共有することができます。KSM はシステムが NUMA を使用していることを検知でき、かつ複数の異なる NUMA ノード間でマージするページを制御できます。

sysfs /sys/kernel/mm/ksm/merge_across_nodes パラメーターを使用して、複数の異なる NUMA ノード間でのページのマージを制御します。デフォルトで、すべてのノードのページは 1 つにマージできます。このパラメーターが 0 (ゼロ) に設定されると同じノードのページのみがマージされます。

通常、システムメモリーをオーバーサブスクライブしない場合、KSM 共有を無効にすることにより、ランタイムパフォーマンスを強化できます。

重要

KSM が複数のゲスト仮想マシンを持つ NUMA ホストの複数のノード間でマージする場合、より離れた場所にあるノードのゲストおよび CPU において、マージされた KSM ページへのアクセスの待ち時間が大幅に増加する可能性があります。

ハイパーバイザーに対してゲストの共有ページを無効にするよう指示するには、以下をゲストの XML に追加します。

```
<memoryBacking>
        <nosharepages/>
</memoryBacking>
```

<memoryBacking> 要素を使用してメモリー設定を調整する方法についての詳細は、「virsh を使用したメモリーチューニング」を参照してください。

付録A 改訂履歴

改訂 1.0-35.1	Mon Jan 4 2021	Red Hat
翻訳ファイルを XML ソースバージョン 1.0-35 と同期		
改訂 1.0-35	Thus May 23 2019	Jiri Herrmann
7.7 ベータ版公開用バージョン		
改訂 1.0-34	Tue Oct 25 2018	Jiri Herrmann
7.6 GA 公開用バージョン		
改訂 1.0-32	Tue Aug 14 2018	Jiri Herrmann
7.6 ベータ版公開用バージョン		
改訂 1.0-31	Wed Apr 4 2018	Jiri Herrmann
7.5 GA 公開用バージョン		
改訂 1.0-27	Mon Jul 27 2017	Jiri Herrmann
7.4 GA 公開用バージョン		
改訂 1.0-24	Mon Oct 17 2016	Jiri Herrmann
7.3 GA 公開用バージョン		
改訂 1.0-22	Mon Dec 21 2015	Laura Novich
バグ修正後の本書の再発行		
改訂 1.0-19	Thu Oct 08 2015	Jiri Herrmann
改訂履歴の処理		